

国际信息工程先进技术译丛

WILEY

全面详解LTE：MATLAB建模、仿真与实现

Understanding LTE with MATLAB: From Mathematical Modeling to Simulation and Prototyping

[美] Houman Zarrinkoub 著

武 冀 译

Understanding LTE with MATLAB®

From Mathematical Modeling to Simulation and Prototyping

Houman Zarrinkoub

WILEY

◎第一本LTE与MATLAB完美结合的图书，以实用、高效、直观的方式详解LTE

◎详解LTE物理层及相关协议，包含MATLAB源代码，从事LTE领域，你会需要本书

◎MathWorks公司通信和LTE软件工具负责人权威之作

◎MATLAB中文论坛鼎力支持为读者提供全面、超值的配套服务





机械工业出版社
CHINA MACHINE PRESS

国际信息工程先进技术译丛

全面详解 LTE：MATLAB 建模、仿真与实现

[美] Houman Zarrinkoub 著

武 冀 译



机 械 工 业 出 版 社

本书通过关键核心技术的理论概览、简明扼要地讨论 LTE 标准规范和用于仿真 LTE 标准所需的 MATLAB 算法这三个部分审视了 LTE 标准中的物理层，并通过一系列的程序，展现了每一种 LTE 的核心技术，通过一步步综合这些核心技术，最终建立 LTE 物理层的系统模型并评价系统性能。通过这一循序渐进的过程，读者将会在仿真中深入理解 LTE 的技术构思和标准规范。

本书适合通信、电子工程和计算机专业的学生、研究者和教授阅读，也可供通信系统的工程师、设计师和配置人员参考。

北京市版权局著作权合同登记图字：01 - 2014 - 5108 号

图书在版编目（CIP）数据

全面详解 LTE：MATLAB 建模、仿真与实现/（美）扎林克伯（Zarrinkoub, H.）著；武冀译. —北京：机械工业出版社，2015.3

（国际信息工程先进技术译丛）

书名原文：Understanding LTE with MATLAB: from mathematical modeling to simulation and prototyping

ISBN 978-7-111-48919-1

Ⅰ.①全… Ⅱ.①扎…②武… Ⅲ.①无线电通信-移动网-研究 Ⅳ.①TN929.5

中国版本图书馆 CIP 数据核字（2014）第 296417 号

机械工业出版社（北京市百万庄大街 22 号 邮政编码 100037）

策划编辑：林 桢 责任编辑：吕 潇

责任校对：张 薇 封面设计：马精明

责任印制：乔 宇

北京机工印刷厂印刷（三河市南杨庄国丰装订厂装订）

2015 年 4 月第 1 版第 1 次印刷

169mm × 239mm · 28.75 印张 · 529 千字

0 001—3 000 册

标准书号：ISBN 978-7-111-48919-1

定价：88.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
服务咨询热线：(010)88361066	机 工 官 网：www.cmpbook.com
读者购书热线：(010)68326294	机 工 官 博：weibo.com/cmp1952
(010)88379203	教育服务网：www.cmpedu.com
封面无防伪标均为盗版	金 书 网：www.golden-book.com

推 荐 序 一

This book, "Understanding LTE with MATLAB" By Dr. Houman Zarrinkoub, is a gem among a large number of books on LTE technology. This is an assessment shared by many readers who, by reading this book, have mastered the theory and concept of LTE technology through hand – on simulation of MATLAB programs. There are 3 unique benefits of the book that make this book truly stand out.

The first unique benefit of this book is the collection of MATLAB examples and test benches that come as essential parts of this book. Through an intuitive and pedagogic approach, the book builds up components of the LTE physical layer (PHY) progressively from simple to more complex using MATLAB programs. These MATLAB programs in the book not only serves as training material on LTE and LTE – Advanced technology, but also empowers readers to obtain a deeper understanding of the subject matter through simulation of MATLAB programs. The MATLAB programs cover simulation, verification and implementation of various components of the LTE system, as well as practical modeling and simulation of the entire LTE system. The up – to – date MATLAB source code can be obtained from the publisher's web site.

The second unique benefit is that the book explains the complex and hard-to-understand PHY specification of the LTE standards in a simple and easy-to-understand approach. The book strikes a well-thought balance in covering the theoretical background of the enabling technologies, the details regarding the standard specifications, and the algorithms and simulation test benches needed to implement the design. The book focuses on the most common features of the PHY model, and provides an overview treatment of enabling technologies of standard. This helps dissolve the mystery surrounding the LTE PHY specification and makes the book easy to read. Coupled with simulation models of the LTE standard, this book helps readers quickly develop an understanding of the elements that make up a communications system, and obtain a programmatic recipe for the sequence of operations that make up the PHY specifications.

The third unique benefit of this book is that the book focuses specifically on the LTE standard and its evolution and targets a diverse set of readers from different backgrounds: algorithm designers and system engineers working in large and small telecommunications companies, professors and researchers who are working on innovative new

designs for wireless systems, and masters and Ph. D students who want to put together a PHY model of the LTE standard and use those simulations to test their innovative new ideas and designs. Unlike many titles that treat only the mathematical foundation of the standard, this book discusses the mathematical formulation of many enabling technologies (such as OFDM and MIMO) in the context of the overall performance of the system. Furthermore, by including chapters dedicated to simulation, performance evaluation, and implementation, the book broadens its appeal to a much larger readership composed of both academicians and practitioners.

In a recent conversation with me, Dr. Zarrinkoub expressed his appreciation for the enthusiastic correspondences with the readers. He is looking forward to more discussions with his readers after the release of the Chinese edition, and is always ready to make improvement to the book and MATLAB programs, hoping the book will provide a positive influence in every reader's professional lives.

John Zhao

Worldwide product marketing manager, MathWorks

推荐序一（译文）

Houman Zarrinkoub 博士撰写的这本书，《全面详解 LTE：MATLAB 建模、仿真与实现》，是大量关于 LTE 技术方面的书籍中的一块瑰宝，这是众多受益读者的共同评价。很多读者已经通过运行书中提供的 MATLAB 程序迅速地掌握了 LTE 通信技术的理论和概念。本书之所以能够脱颖而出是因为本书具有三个优势。

第一个优势是，书中贯穿着大量的 MATLAB 实例和测试平台的源代码。本书结合直观理解和理论教学的方法，用 MATLAB 程序帮读者由浅入深地逐步建立起 LTE 物理层（PHY）的每个组件。这些 MATLAB 程序不仅可以作为 LTE 和 LTE - A 的实用培训教材，而且还可以通过仿真 MATLAB 程序帮读者加深对 LTE 理论层面的理解。书中提供的 MATLAB 程序包括仿真，验证和实现 LTE 系统的各种组件，也包括整个 LTE 系统的实际建模和仿真。最新 MATLAB 源代码可以从出版商的网站上获得。

第二个优势是，本书用浅显易懂的办法来讲解复杂深奥的 LTE 物理层的规范。本书既涵盖了相关的无线通信的理论背景，也深入探讨了 LTE 标准规范的细节，同时也提供了具体的算法和仿真测试平台。本书的重点集中在物理层的最常见的功能上，同时也对涉及的无线通信的基础理论进行了概括。这样有助于揭开 LTE 物理层规范的神秘面纱，使本书易于阅读。结合书中提供的 LTE 的仿真模型，本书可以帮助读者迅速理解 LTE 通信系统中的每一个组件，并很快对 LTE 物理层具体操作形成全面的认识。

第三个优势是，本书的主要内容集中在介绍 LTE 标准的内容和它的演变过程上。本书面向来自不同领域，具有不同技术背景的读者：大型或小型的电信公司的算法设计人员、系统工程师、无线系统研究领域的教授和研究人员，以及硕士和博士研究生。他们可以建立起 LTE 标准的物理层的模型，通过数字模拟来测试和验证其理论和设计上的创新。不像其他只注重通信标准的数学基础的书，本书着手于 LTE 的关键技术（如 OFDM 和 MIMO）的数学公式和演变，以及这些关键技术对系统的整体性能的影响。此外，由于书中包含了这些系统模拟和系统性能评估的专门章节，本书对学术界和企业界的读者都同样适用。

在我们最近的一次谈话中，Zarrinkoub 博士对为本书提出宝贵意见的热心读者们表示衷心感谢。他期待在本书中文版发行后，他会与更多的中文读者进行交

流和讨论，并随时听取读者的建议，对书中的内容和MATLAB程序进行改进。他希望这本书能对每一位读者的职业生涯产生积极的影响。

赵志宏（John Zhao）
MathWorks美国总部产品市场经理

推 荐 序 二

LTE 和 LTE - Advance 是由 3GPP 开发的新一代移动通信标准。伴随着我国、北美、欧洲、日本覆盖 LTE 网络之后，世界电信业正式拉开了 4G 网络运营的帷幕。

本书中，作者用 MATLAB 这个科学运算语言与仿真环境，清晰地解释了 LTE 技术的数学思想与架构，详细解读了 LTE 移动通信标准，并聚焦其物理层，构建算法、测试平台等，通过仿真加深读者对技术的理解。这样有利于读者了解 LTE 是如何以及为什么能完成如此标志性的技术成果。

本书还循序渐进深入浅出地介绍了 MATLAB 中通信系统工具箱、数字处理工具箱和系统工具箱 3 个部分的上百种组件，讲解和实现了与 LTE 出色性能相关的关键技术及其数学基础，如正交频分复用（OFDM）、多输入多输出（MIMO）、Turbo 编码和动态链路自适应等技术。真正将 MATLAB 变为易于驾驭的工具，将使用者从研究工具的使用方法中解放出来，将重点放到设计本身，从而将极大地提高设计效率，极大方便了 MATLAB 用户的使用。

本书不但可以帮助对移动通信技术感兴趣的工程师、研究者群体从另一个角度更加深入地理解 LTE 技术标准和关键技术。同时，本书还提供了一个能够更加深入理解 MATLAB 的机会，帮助读者充分发挥 MATLAB 的潜能。读者可以利用 MATLAB 验证、改进和创新，推进未来移动通信系统的研究和进步。

李楠

中国移动通信集团设计院有限公司

译者序

LTE是当前最前沿的移动通信标准，以其为模板的4G网络已经成为世界无线移动通信网络的最新发展方向。我国在北美、欧洲、日本覆盖4G网络之后紧跟技术发展步伐，以东部超大城市圈为试点，逐步将4G推广到全国各大中城市。LTE以及LTE - Advanced技术凭借其向下兼容性的特点，将为设备层和应用层更新提供极大帮助。基于4G技术的各种应用如雨后春笋般出现并蓬勃发展，极大地丰富和满足了社会各群体的日常生活和各种需要。

本书就在这样一个背景下应运而生。本书基于MATLAB这样一个广泛应用于科学计算和建模领域的语言和开发环境，对LTE标准的建模和实现进行深入讨论。本着从实际问题中来，到服务实际应用中去的原则，以最贴近真实移动通信环境的建模为途径，分块剖析了LTE标准的各关键技术和实现方法，为广大通信领域的师生、关注学术前沿的学者、开发团体和现场工程师提供了方法和工具引导。

本书用最基础和易懂的理论知识概括了现代移动通信技术中的重要概念，如OFDM、MIMO以及链路自适应等。同时，本书意留下或指出了可扩展和创新的突破口，可推动LTE和移动通信技术的不断进步，为广大有志于推进移动通信技术创新的科学技术工作者们点燃明灯，影响深远。

MATLAB在科学计算领域以其普适性和精确性为业界所称道。从其问世至今，其始终独占科学计算语言和开发环境的鳌头。但同时，其几乎无所不包的功能组件也带来了使用上的不便。仅移动通信领域应用，就涉及通信系统工具箱、数字处理工具箱和系统工具箱三个部分上百种组件。如何正确使用相应组件构建如LTE这样的现代移动通信系统，是不少MATLAB初学者棘手的问题。本书针对这个问题循序渐进、有的放矢，以若干个关键工具箱组件为基础，以点带面，较为详细地讲解了这些工具箱组件的使用和配置条件以及方法。极大方便了MATLAB用户的使用。真正将MATLAB变为易于驾驭的工具，将使用者从研究工具使用中解放出来，将重点放到设计本身，从而将极大地提高设计效率。

愿这本书如作者所希望的，真正服务学生、老师、研究者和系统设计工程师群体，特别是国内移动通信领域，点燃无线通信技术未来发展创新之火。为更多惊艳和实用技术的出现尽微薄之力！

原书前言

LTE（Long Term Evolution，长期演进）技术和LTE－Advanced（高级LTE）是由3GPP（Third Generation Partnership Project，第三代合作伙伴计划）开发的最新移动通信标准。这两个标准的推出标志着移动通信技术进步中革命性的变革。近十年来，网络架构与移动终端被设计和升级以支持LTE标准。伴随着这些系统扩展到全球各个角落，LTE标准真正实现了全球基带移动接入技术的梦想。

本书将会详细解读LTE移动通信标准，特别是针对它的物理层，以求理解LTE是如何以及为什么能完成如此标志性的技术成果。我们也同时会从学术与实用的角度去审视它。我们将会讲解与实现LTE的出色性能相关的关键技术及其数学基础，比如正交频分复用（OFDM）和多输入多输出（MIMO）。我们也会说明实现这些架构与组件的实用工程学思想。作为本书的组成部分，我们使用MATLAB这个在科学与工程领域广泛应用的科学运算语言与仿真环境，来清晰地解释LTE技术的数学思想与架构，并构建算法、测试平台和图例，希望通过仿真让读者深入理解该技术。

本书针对学术领域与专业工程技术人员编写，明确聚焦LTE标准和它的发展过程。区别于很多书籍只讲解某一个技术标准的数学基础，本书将在文中讲解LTE所涉及的很多技术（比如OFDM和MIMO）。而且，很多章节专注于仿真、性能评估和应用，使本书广泛适用于除学术研究与工程技术人员之外更广阔领域的读者。

本书将用一种直观和便于讲解的方法，使用MATLAB从简单到复杂，一点点构建LTE物理层的组件。通过MATLAB程序仿真，读者将会自信地感到不仅可以完全理解技术标准必要的细节，也可以学会应用它们的能力。

本书力求清晰地阐述LTE物理层模型相关的技术细节，所以通信理论和数字信号处理的基础知识是必备的。电子工程系本科高年级的课程基本涵盖了这些知识。为了通过MATLAB仿真展开教学，MATLAB语言基础也是需要提前掌握的内容。本书既面向电子工程和计算机专业的学生、研究者和教授，也面向通信系统的工程师、设计师和配置人员。这些从技术与编程两方面阐述的知识在日常工作中具有较强应用性。根据读者的领域不同，本书内容大致分为导论、中级和高级三个层次。

本书在构思上可分为两个部分。第一部分结合MATLAB算法对LTE的物理层进行建模，使读者可以仿真和验证系统的各个组件。第二部分结合实际深入详

解诸如系统仿真与配置以及各组件的原型构建等问题。第 1 章，对 LTE 标准进行了概括性的介绍，包括其起源、目标以及四种作为基本组件的关键技术（OFDM、MIMO、Turbo 编码和动态链路自适应原理）；第 2 章快速而充分地概述了 LTE 物理层的规格；第 3 章介绍了本书所使用 MATLAB 和 Simulink 的建模、仿真以及实现的有关功能；第 4 ~7 章深入详解了 LTE 标准的每一个关键技术（调制和编码、OFDM、MIMO 和链路自适应）并建立 MATLAB 模型，互相参照逐步构建基于这些技术的 LTE 物理层的组件；第 8 章总结并深刻分析前文中讲解的关键技术，并展示了 MATLAB 如何对 LTE 的物理层建模，从而对第一部分做了归纳。第 9 章讨论了如何应用多种方案提高 MATLAB 的运行速度，这些方案包括并行计算、C 代码自动生成、GPU 运算、更高效的算法等；第 10 章讨论了一些实现方面的问题，比如目标环境以及它们如何影响编程风格等，作为硬件配置的前提，也讨论了数据的定点数表记法及其对性能的影响；第 11 章总结了前文所有的讨论并做一些前瞻性的研究方向预测。

所有对复杂通信系统如 LTE 技术的背景介绍，都需要设定一定的讲解范围。我们明确了三个概念上的要素，这样可以帮助读者深刻理解 LTE 标准如何工作：

- 关键技术的理论背景；
- 标准规范详细内容；
- 实现设计的算法与仿真验证手段。

为了更有效地扩展本书的内容，本书决定打破对每一个方面都采用平均深度讲解的套路，而是选择对理论基础和标准的技术规范进行充分讨论，并对实现 LTE 标准的多种模式的算法与验证手段进行更全面详细的讲解，覆盖开发 MATLAB应用所需的专业技术。下面两个因素促使本书进行这样的选择：

● 很多书籍对上面第一和第二个方面进行了非常全面的讲解，但并没有关注算法与仿真，强调仿真是本书的新意；

● 通过提供 LTE 标准的仿真模型，我们希望读者扩展理解这些组成通信系统的模型组件和它们内含的编程技巧，从而整合成整个 LTE 物理层模型的一系列操作。算法与验证手段也会通过仿真过程自然地揭示系统的动态特性。

从这个意义上讲，对仿真细节的深入洞悉和理解是非常宝贵的，这会使读者精通自己研究的问题。更珍贵的是，它们使读者有自信去尝试新想法、新提议和验证新的改进，以及利用新工具和模型帮助大学生从理论知识迈向实践，从而最终获得创新、设计、应用的能力。

希望本书可以为对移动通信感兴趣的年轻研究者、学生、教授群体提供一个有现实意义的建模和仿真 LTE 标准的框架。也希望读者能分享本书的所学所得，提出改进和创新意见，推动未来移动通信系统的研究和发展。

专业词汇缩略语表

ASIC	Application-Specific Integrated Circuit	专用集成电路
BCH	Broadcast Channel	广播信道
BER	Bit Error Rate	误码率
BPSK	Binary Phase Shift Keying	二进制移相键控
CP	Cyclic Prefix	循环前缀
CQI	Channel Quality Indicator	信道质量指示符
CRC	Cyclic Redundancy Check	循环冗余检查
CSI	Channel State Information	信道状态信息
CSI－RS	Channel State Information Reference Signal	信道状态信息参考信号
CSR	Cell-Specific Reference	小区专有参考
CUDA	Compute Unified Device Architecture	统一计算设备架构
DM－RS	Demodulation Reference Signal	解调参考信号
DSP	Digital Signal Processor	数字信号处理器
eNodeB	enhanced Node Base station	增强型基站
E－UTRA	Evolved Universal Terrestrial Radio Access	演进通用陆地无线接入
FDD	Frequency Division Duplex	频分双工
FPGA	Field-Programmable Gate Array	现场可编程门阵列
HARO	Hybrid Automatic Repeat Request	混合自动重传请求
HDL	Hardware Description Language	硬件描述语言
IMT	International Mobile Telecommunication	国际移动通信
LTE	Long Term Evolution	长期演进
MAC	Medium Access Control	媒介访问控制
MBMS	Multimedia Broadcast and Multicast Service	多媒体广播组播业务
MBSFN	Multicast/Broadcast over Single Frequency Network	组播/广播单频网络
MIMO	Multiple Input Multiple Output	多输入多输出
MMSE	Minimum Mean Square Error	最小均方误差
MRC	Maximum Ratio Combining	最大比合并

MU – MIMO	Multi-User Multiple Input Multiple Output	多用户多输入多输出
OFDM	Orthogonal Frequency Division Multiplexing	正交频分复用
PBCH	Physical Broadcast Channel	物理广播信道
PCFICH	Physical Control Format Indicator Channel	物理控制格式指示信道
PCM	Pulse Code Modulation	脉冲编码调制
PDCCH	Physical Downlink Control Channel	物理下行链路控制信道
PDSCH	Physical Downlink Shared Channel	物理下行链路共享信道
PHICH	Physical Hybrid ARQ Indicator Channel	物理 HARQ 指示信道
PHY	Physical Layer	物理层
PMCH	Physical Multicast Channel	物理组播信道
PRACH	Physical Random Access Channel	物理随机接入信道
PSS	Primary Synchronization Signal	主同步信号
PUCCH	Physical Uplink Control Channel	物理上行链路控制信道
PUSCH	Physical Uplink Shared Channel	物理上行链路公共信道
QAM	Quadrature Amplitude Modulation	正交振幅调制
QPP	Quadratic Permutation Polynomial	二次置换多项式
QPSK	Quadrature Phase Shift Keying	正交移相键控
RLC	Radio Link Control	无线链路控制协议
RMS	Root Mean Square	方均根
RRC	Radio Resource Control	无线资源控制协议
RTL	Register Transfer Level	寄存器传输级
SC – FDM	Single-Carrier Frequency Division Multiplexing	单载波频分复用
SD	Sphere Decoder	球形译码器
SFBC	Space-Frequency Block Coding	空频分组编码
SINR	Signal-to-Interference-plus-Noise Ratio	信号与干扰加噪声比
SNR	Signal-to-Noise Ratio	信噪比
SSD	Soft-Sphere Decoder	软球形译码器

SSS	Secondary Synchronization Signal	辅助同步信号
STBC	Space-Time Block Coding	空时分组编码
SFBC	Space-Frequency Block Coding	空频分组编码
SU－MIMO	Single-User MIMO	单用户 MIMO
TDD	Time-Division Duplex	时分双工
UE	User Equipment	用户设备
ZF	Zero Forcing	迫零算法

目　录

推荐序一
推荐序一（译文）
推荐序二
译者序
原书前言
专业词汇缩略语表
1　导论 ………………………………… 1
1.1　无线通信标准速览 ……………… 1
1.2　数据速率的历史 ………………… 4
1.3　IMT - Advanced 要求………………… 4
1.4　3GPP 和 LTE 标准化………………… 5
1.5　LTE 要求 ………………………… 5
1.6　理论策略 ………………………… 6
1.7　LTE 关键技术 ………………………… 7
1.7.1　OFDM ……………………… 7
1.7.2　SC - FDM ………………… 8
1.7.3　MIMO ……………………… 8
1.7.4　Turbo 信道编码………………… 8
1.7.5　链路自适应 ……………… 9
1.8　LTE 物理层建模 ……………… 9
1.9　LTE（R8 版和 R9 版） ………… 10
1.10　LTE - Advanced（R10 版） …… 11
1.11　MATLAB 和无线系统设计 …… 11
1.12　本书组织结构 ………………… 11
参考文献 ……………………………… 12
2　LTE 物理层概览 ………………… 13
2.1　空中接口 …………………………… 13
2.2　频带 ………………………………… 14
2.3　单播和组播服务 ………………… 15
2.4　带宽分配 ………………………… 16
2.5　时间帧 …………………………… 17
2.6　时 - 频分布 ………………………… 18
2.7　OFDM 多载波传输 ……………… 20
2.7.1　循环前缀 ………………… 20
2.7.2　子载波间隔 ……………… 21
2.7.3　资源块尺寸 ……………… 21
2.7.4　频域调度 ………………… 22
2.7.5　接收端典型操作 ………… 22
2.8　单载波频分复用 ……………… 23
2.9　资源网格的内容 ……………… 23
2.10　物理信道 ……………………… 24
2.10.1　下行链路物理信道 ……… 25
2.10.2　下行链路信道功能 ……… 26
2.10.3　上行链路物理信道 ……… 29
2.10.4　上行链路信道功能 ……… 30
2.11　物理信号 ……………………… 30
2.11.1　参考信号 ………………… 30
2.11.2　同步信号 ………………… 32
2.12　下行链路帧结构 ……………… 32
2.13　上行链路帧结构 ……………… 33
2.14　MIMO ……………………………… 34
2.14.1　接收分集 ………………… 34
2.14.2　发射分集 ………………… 34
2.14.3　空分复用 ………………… 36
2.14.4　波束赋形 ………………… 37
2.14.5　循环延迟分集 …………… 38
2.15　MIMO 模式……………………… 38
2.16　物理层数据处理 ……………… 39
2.17　下行链路数据处理 …………… 39
2.18　上行链路数据处理 …………… 40
2.18.1　SC - FDM ………………… 41
2.18.2　MU - MIMO ……………… 42
2.19　本章小结 ……………………… 43
参考文献 ……………………………… 43
3　MATLAB 通信系统设计 ……… 44

3.1 系统开发流程 …… 44
3.2 挑战和能力 …… 44
3.3 关注点 …… 45
3.4 目标 …… 45
3.5 MATLAB 的物理层模型 …… 46
3.6 MATLAB …… 46
3.7 MATLAB 工具箱 …… 47
3.8 Simulink 组件 …… 47
3.9 建模与仿真 …… 48
3.9.1 DSP 系统工具箱 …… 48
3.9.2 通信系统工具箱 …… 48
3.9.3 并行计算工具箱 …… 49
3.9.4 定点型设计器 …… 49
3.10 原型建模与实现 …… 49
3.10.1 MATLAB 代码生成器 …… 50
3.10.2 硬件实现 …… 50
3.11 系统对象介绍 …… 51
3.11.1 通信系统工具箱的系统对象 …… 51
3.11.2 系统对象的测试平台 …… 52
3.11.3 系统对象函数 …… 54
3.11.4 字符误码率仿真 …… 56
3.12 MATLAB 信道编码实例 …… 57
3.12.1 纠错与检错 …… 57
3.12.2 卷积码 …… 58
3.12.3 硬判决 Viterbi 译码 …… 58
3.12.4 软判决 Viterbi 译码 …… 60
3.12.5 Turbo 编码 …… 62
3.13 本章小结 …… 64
参考文献 …… 65
4 调制和编码 …… 66
4.1 LTE 调制方案 …… 66
4.1.1 MATLAB 实例 …… 68
4.1.2 BER 测量 …… 72
4.2 比特级绕码 …… 74
4.2.1 MATLAB 实例 …… 75
4.2.2 BER 测量 …… 78
4.3 信道编码 …… 79
4.4 Turbo 编码 …… 79
4.4.1 Turbo 编码器 …… 80
4.4.2 Turbo 译码器 …… 81
4.4.3 MATLAB 实例 …… 81
4.4.4 BER 测量 …… 83
4.5 早期终止机制 …… 87
4.5.1 MATLAB 实例 …… 87
4.5.2 BER 测量 …… 88
4.5.3 计时测量 …… 91
4.6 码率匹配 …… 91
4.6.1 MATLAB 实例 …… 92
4.6.2 BER 测量 …… 95
4.7 码块分段 …… 97
4.7.1 MATLAB 实例 …… 97
4.8 LTE 传输信道处理 …… 99
4.8.1 MATLAB 实例 …… 99
4.8.2 BER 测量 …… 101
4.9 本章小结 …… 103
参考文献 …… 103
5 OFDM …… 104
5.1 信道建模 …… 104
5.1.1 大尺度和小尺度衰落 …… 104
5.1.2 多径衰落效应 …… 105
5.1.3 多普勒效应 …… 105
5.1.4 MATLAB 实例 …… 105
5.2 讨论范围 …… 110
5.3 工作流程 …… 110
5.4 OFDM 和多径衰落 …… 110
5.5 OFDM 和信道响应估计 …… 111
5.6 频域均衡 …… 112
5.7 LTE 资源网格 …… 112
5.8 配置资源网格 …… 114
5.8.1 CSR 符号 …… 114
5.8.2 DCI 符号 …… 115
5.8.3 BCH 符号 …… 115
5.8.4 同步符号 …… 116
5.8.5 用户数据符号 …… 116
5.9 参考信号生成 …… 118

5.10 资源元素映射 …………………… 120
5.11 OFDM 信号生成 ………………… 124
5.12 信道建模 …………………………… 125
5.13 OFDM 接收端 ……………………… 127
5.14 资源元素反映射 …………………… 129
5.15 信道估计 …………………………… 131
5.16 均衡器增益计算 …………………… 133
5.17 信道可视化 ………………………… 134
5.18 下行链路传输模式 1 ……………… 135
5.18.1 SISO 模型 …………………… 135
5.18.2 SIMO 模型 …………………… 142
5.19 本章小结 …………………………… 150
参考文献 ……………………………………… 151
6 MIMO ……………………………………… 152
6.1 MIMO 定义 ………………………… 152
6.2 MIMO 的动机 ……………………… 153
6.3 MIMO 的种类 ……………………… 153
6.3.1 接收端合并技术 ……………… 153
6.3.2 发射分集 ……………………… 154
6.3.3 空分复用 ……………………… 154
6.4 MIMO 的覆盖范围 ………………… 154
6.5 MIMO 信道 ………………………… 154
6.5.1 MATLAB 实现 ……………… 155
6.5.2 LTE 特征信道模型 ………… 157
6.5.3 MATLAB 实现 ……………… 159
6.5.4 MIMO 信道初始化 …………… 160
6.5.5 添加 AWGN ………………… 161
6.6 MIMO 的一般特征 ………………… 161
6.6.1 MIMO 资源网格结构 ………… 162
6.6.2 资源元素映射 ………………… 163
6.6.3 资源元素反映射 ……………… 166
6.6.4 基于 CSR 的信道估计 ……… 170
6.6.5 信道估计函数 ………………… 171
6.6.6 信道估计扩展 ………………… 173
6.6.7 理想信道估计 ………………… 177
6.6.8 信道响应提取 ………………… 179
6.7 MIMO 的特殊特征 ………………… 180
6.7.1 发射分集 ……………………… 180
6.7.2 收发器启动函数 ……………… 188
6.7.3 下行链路传输模式 2 ……… 197
6.7.4 空分复用 ……………………… 204
6.7.5 空分复用中的 MIMO 操作 ……………………… 207
6.7.6 下行链路传输模式 4 ……… 215
6.7.7 开环空分复用 ……………… 229
6.7.8 下行链路传输模式 3 ……… 233
6.8 本章小结 …………………………… 240
参考文献 ……………………………………… 241
第 7 章 链路自适应 ………………… 242
7.1 系统模型 …………………………… 243
7.2 LTE 中的链路自适应 ……………… 244
7.2.1 信道质量估计 ……………… 244
7.2.2 预编码矩阵估计 …………… 245
7.2.3 秩估计 ……………………… 245
7.3 MATLAB 实例 …………………… 245
7.3.1 CQI 估计 …………………… 245
7.3.2 PMI 估计 …………………… 248
7.3.3 RI 估计 ……………………… 249
7.4 子帧间的链路自适应 ……………… 252
7.4.1 收发端模型结构 …………… 253
7.4.2 更新收发端参数结构体 ……………………… 254
7.5 自适应调制 ………………………… 255
7.5.1 无自适应 …………………… 255
7.5.2 随机变更调制方案 ………… 256
7.5.3 基于 CQI 的自适应 ……… 256
7.5.4 收发端性能验证 …………… 257
7.5.5 结论 ………………………… 259
7.6 自适应调制与编码率 ……………… 260
7.6.1 无自适应 …………………… 260
7.6.2 随机变更调制方案 ………… 261
7.6.3 基于 CQI 的自适应 ……… 261
7.6.4 收发端性能验证 …………… 262
7.6.5 结论 ………………………… 262
7.7 自适应预编码 ……………………… 264
7.7.1 基于 PMI 的自适应 ……… 266

7.7.2 收发端性能验证 ………… 267
7.7.3 结论 ………………………… 268
7.8 自适应 MIMO ………………… 268
7.8.1 基于 RI 的自适应 ………… 270
7.8.2 收发端性能验证 ………… 271
7.8.3 结论 ………………………… 272
7.9 下行链路控制信息 ………… 272
7.9.1 MCS ………………………… 272
7.9.2 自适应率 ………………… 275
7.9.3 DCI 处理 ………………… 275
7.10 本章小结 ……………………… 279
参考文献 ……………………………… 280
8 系统级建模 …………………… 281
8.1 系统模型 ……………………… 281
8.1.1 发射端模型 ……………… 282
8.1.2 发射端模型的 MATLAB 模型 ……………………… 283
8.1.3 信道模型 ………………… 285
8.1.4 信道模型的 MALTAB 模型 ……………………… 285
8.1.5 接收端模型 ……………… 286
8.1.6 接收端模型的 MATLAB 模型 ……………………… 287
8.2 用 MATLAB 构建的系统模型 ……………………………… 289
8.3 定量评估 ……………………… 291
8.3.1 传输模式的影响 ………… 291
8.3.2 BER 与 SNR 的函数关系 ……………………… 293
8.3.3 信道估计技术的影响 …… 295
8.3.4 信道模型的影响 ………… 295
8.3.5 信道时延扩散与循环前缀的影响 ……………… 296
8.3.6 MIMO 接收器算法的影响 ……………………… 297
8.4 吞吐量分析 …………………… 298
8.5 用 Simulink 进行系统建模 …… 299
8.5.1 构建一个 Simulink 模型 … 301
8.5.2 Simulink 集成 MATLAB 算法 ……………………… 302
8.5.3 参数初始化 ……………… 309
8.5.4 运行仿真 ………………… 311
8.5.5 引入参数对话框 ………… 313
8.6 定量评估 ……………………… 321
8.6.1 声音信号传输 …………… 321
8.6.2 主观声音质量测试 ……… 322
8.7 本章小结 ……………………… 323
参考文献 ……………………………… 323
9 仿真 ……………………………… 324
9.1 提升 MATLAB 仿真速度 ……… 324
9.2 工作流程 ……………………… 325
9.3 实例研究：LTE PDCCH 处理 ……………………………… 326
9.4 基准算法 ……………………… 327
9.5 MATLAB 代码剖析 …………… 329
9.6 MATLAB 代码优化 …………… 331
9.6.1 向量化 …………………… 331
9.6.2 预分配 …………………… 337
9.6.3 系统对象 ………………… 340
9.7 使用加速功能 ………………… 351
9.7.1 MATLAB—C 代码生成 …… 352
9.7.2 并行运算 ………………… 353
9.8 使用 Simulink 模型 ………… 355
9.8.1 创建 Simulink 模型 ……… 355
9.8.2 验证数值等价性 ………… 356
9.8.3 Simulink 基准模型 ……… 357
9.8.4 优化 Simulink 模型 ……… 358
9.9 GPU 辅助运算 ……………… 366
9.9.1 在 MATLAB 中启动 GPU 功能 ……………………… 367
9.9.2 GPU 优化系统对象 ……… 367
9.9.3 使用单一 GPU 系统对象 ……………………… 368
9.9.4 GPU 参与并行计算 ……… 370
9.10 实例研究：在 GPU 上进行 Turbo 编码 ……………………… 374

9.10.1 基于 CPU 处理基准算法 …… 374
9.10.2 基于 GPU 处理 Turbo 译码器 …… 377
9.10.3 基于 GPU 处理多个系统对象 …… 378
9.10.4 多帧和大数据长度 …… 380
9.10.5 使用单精度数据类型 …… 383
9.11 本章小结 …… 385
10 基于 C/C++代码的原型构建 …… 387
10.1 应用范围 …… 387
10.2 动机 …… 388
10.3 要求 …… 388
10.4 MATLAB 代码的构思 …… 389
10.5 如何创建代码 …… 389
10.5.1 实例研究：频域均衡 …… 389
10.5.2 使用 MATLAB 命令 …… 390
10.5.3 使用 MATLAB 代码转换器工程 …… 392
10.6 转换的 C 代码的结构 …… 397
10.7 支持的 MATLAB 子集 …… 398
10.7.1 代码转换准备 …… 398
10.7.2 实例研究：插入导频信号 …… 399
10.8 复数和本地 C 类型 …… 400
10.9 系统工具箱支持 …… 403
10.9.1 实例研究：FFT 和反 FFT …… 403
10.10 定点型数据支持 …… 408
10.10.1 实例研究：FFT 函数 …… 409
10.11 可变长度数据支持 …… 412
10.11.1 实例研究：自适应性调制 …… 412
10.11.2 定长代码转换 …… 413
10.11.3 有界变长数据 …… 417
10.11.4 无界变长数据 …… 419
10.12 集成外部 C/C++代码 …… 421
10.12.1 算法 …… 421
10.12.2 执行 MATLAB 测试平台 …… 423
10.12.3 生成 C 代码 …… 425
10.12.4 接口函数 C 代码 …… 426
10.12.5 主函数 C 代码 …… 429
10.12.6 编译和连接 …… 430
10.12.7 执行 C 测试平台 …… 432
10.13 本章小结 …… 433
参考文献 …… 433
11 总结 …… 434
11.1 建模 …… 434
11.1.1 理论构思 …… 434
11.1.2 标准规范 …… 435
11.1.3 MATLAB 算法 …… 435
11.2 仿真 …… 436
11.2.1 仿真加速 …… 437
11.2.2 加速方法 …… 437
11.2.3 实现 …… 437
11.3 未来工作的方向 …… 438
11.3.1 用户层面 …… 438
11.3.2 控制层面处理 …… 439
11.3.3 混合自动重传请求 …… 439
11.3.4 系统接入模型 …… 439
11.4 结语 …… 440
译后记 …… 441

1 导　　论

我们生活在一个移动数据革命的年代。随着智能手机、便携式计算机和平板电脑大规模的市场扩展，移动通信系统提供的用户需求服务和应用已经远远不止声音通话。像网页浏览、网上社交网络以及音乐和视频流媒体这些密集数据量移动服务，逐渐成为推动下一代无线移动通信标准发展的力量。这些新标准拥有了所必需的大数据通信速率和网络能力，以支撑全球化传输这些类型丰富的多媒体应用。

LTE（Long Term Evolution，长期演进）和 LTE - Advance（高级 LTE）标准，迎合了时代需求，实现了全球基带移动通信的愿景。这个愿景与实现它的演进基站包括更高的无线接入数据传输率、更强大的系统运能和覆盖能力、更灵活的带宽操作和显著提高的频谱利用效率；同时，它有低延迟，低运营成本，多路天线支持，以及互联网与现存移动通信系统的无缝继承特点。

从某些角度来说，LTE 和 LTE - advance 作为被大家所熟知的第四代（4G）无线通信系统的代表，可以被认为是其前辈的第三代（3G）无线通信系统的有机升级。另一方面，在其底层传输技术方面，它们从过去技术中颠覆性的飞跃，展现了未来技术发展的一个新的黎明。为了我们承上启下介绍移动技术演化以引出 LTE 标准，我们将会对无线通信标准做一个简短的回顾。在这个回顾里，我们寻根溯源众多在 LTE 标准中的关键技术，阐明它们一些表现在超越早期技术方面的要求。

1.1 无线通信标准速览

在过去的 20 年中，我们曾体验过众多移动通信标准，从 2G 到 3G 到最新的 4G，我们期待这个趋势一直继续（见图 1.1）。2G 标准的主要任务是提供移动通话与声音应用。3G 标准开始了基于数据组的革命，支持如电子邮件，网页浏览和其他客户 - 伺服服务之类的网络应用。4G 标准将会是全 IP 数据组网络，并满足如移动视频点播这样高带宽应用的爆炸性需求。

历史上，那些移动通信标准由北美、欧洲和世界上其他地区的网络供应商和经营商分别制定。第二代（2G）数字移动通信系统最早出现在 1990 年之前。2G 系统主要基于电路交换数据通信技术。欧洲的 GSM（Global System for Mobile Communications，全球移动通信系统）和北美的 IS - 54 都是第一批 2G 标准。这

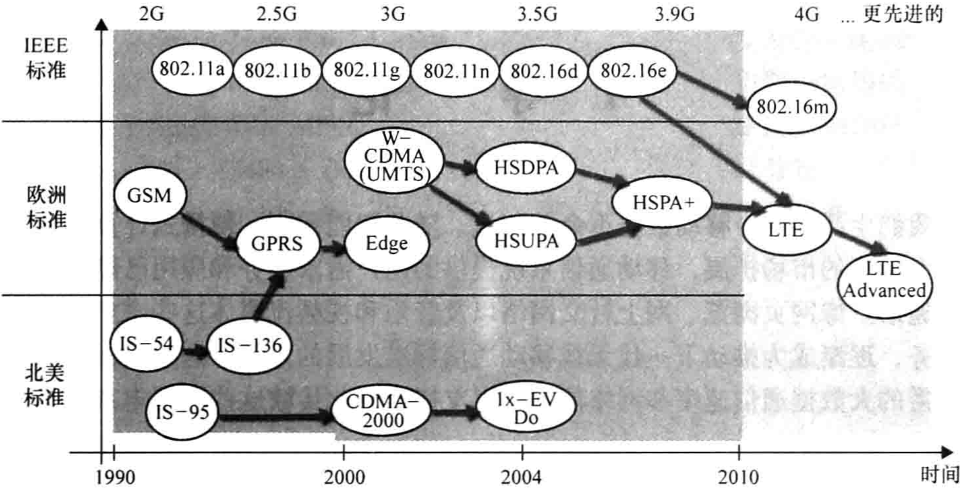


图 1.1　20 年以来无线通信标准的演进

两个标准都是基于时分多址（TDMA）技术。时分多址技术中，一个窄带通信信道被分为若干个时间片，多用户共享所分配时间片的频谱。关于数据速率，如 GSM 系统，它可支持超过 13kbit/s 的声音服务和超过 9.6kbit/s 的数据服务。

GSM 标准随后发展衍生出 GPRS（Generalized Packet Radio Service，通用分组无线服务技术），支持 171.2kbit/s 的峰值数据速率。GPRS 标准标志着核心分离无线网络的出现，它拥有基于组接入的交换技术支持数据传输，以及基于电路交换技术支持声音信号传输。GPRS 技术在后来又进化为 EDGE（Enhanced Data Rates for Global Evolution，全球化增强数据速率演进），引入更高数据传输率调制方案（8PSK，移相键控）和高达 384kbit/s 的峰值数据速率。

在北美，IS－95 是第一个码分多址技术（Code Division Multiple Access，CDMA）的商业开发。CDMA 在 IS－95 标准中体现为应用直接序列扩频技术，基于正交扩频码，支持多用户共享更宽的带宽。IS－95 提供 1.2284MHz 带宽，允许单蜂窝终端最大 64 路声音信道通信和每信道 14.4kbit/s 的峰值数据通信速率。该标准的改进版 IS－95－B 支持基于组接入的高速度数据传输，结合新的扩充编码信道，IS－95－B 支持峰值数据通信速率 115.2kbit/s 的高速组数据传输。在北美，标准化组织 3GPP2（Third Generation Partnership Project 2，第三代合作伙伴计划 2）构建的 3G 系统技术规范和标准就是基于 CDMA 的技术演进。从 1997 年到 2003 年，3GPP2 开发了一系列的基于 IS－95 的标准，包括 1xRTT、1x－EV－DO（Evolved Voice Data Only），和 EV－DV（Evolved Data and Voice）。通过附加 64 路更多的冲突信道，1xRRT 的容量达到 IS－95 的一倍，拥有 307kbit/s 峰值通信速率。通过引入包括自适应性调制和编码、HARQ，turbo 编码，以及基于

更小帧尺寸的更快速规划等技术，1x - EV - DO 和 1x - EV - DV 标准达到了 2.4 ~3.1Mbit/s 范围的峰值通信速率。

3GPP（Third Generation Partnership Project，第三代合作伙伴计划）是最初管理欧洲移动通信标准的技术标准化组织，后来成为全球技术标准化组织。它主要负责 3G 移动系统及其后继者们的技术规范构建。在 1997 年，3GPP 开始努力标准化 ITU IMT - 2000（International Telecommunications Union International Mobile Telecommunication）项目的工作。这个项目的目标是实现将基于 2G TDMA 的 GSM 技术，向基于 CDMA 的 3G 宽带技术转化，称为“通用移动电信系统（Universal Mobile Telecommunications System，UMTS）”。UMTS 在当时标志了移动通信领域的重大改变。它在 2001 年完成标准化并称为 3GPP 协议第 4 个发布版本。UMTS 系统可以实现下行链路峰值通信速率 1.92Mbit/s。UMTS 系统的升级版，HSDPA（High - speed Downlink Packet Access，高速率下行链路分组接入），作为 3G 标准的第 5 个发布版本在 2002 年完成标准化。该标准通过应用更短子帧的快速规划和 16QAM（Quadrature amplitude Modulation，正交幅度调制）调制方案，可提供 14.4Mbit/s 峰值通信速率。告诉下行链路组接入标准化于 2004 年，作为 3G 标准的第 5 个发布版本可提供最大 5.76Mbit/s 的峰值通信速率。这两个标准，加上熟知的 HSPA（高速组接入），随后升级为 3GPP 标准的第 7 个发布版本——HSPA +，或称为 MIMO（多输入多输出）HSDPA。HSPA + 标准可以达到 84Mbit/s 速率。它也是第一个包括 2 ×2 MIMO 技术和更高调制方案（64QAM）的移动通信标准。它也吸收了一部分北美 3G 标准的先进特性。这些特性包括自适应调制和编码，HARQ，turbo 编码，和更快的规划。

另一个驱使实现高速通信速率和频谱效率的重要的无线应用是无线本地网络（Wireless Local Area Network，WLAN）。WLAN 标准的主要用途是为在建筑里的固定用户（家庭和办公室）提供稳定告诉的网络连接。在国际移动通信网络一步一步进化的同时，IEEE（Institute of Electrical and Electronics Engineers，电器电子工程师协会）推进 WLAN 和无线城市网络（WMAN）的标准化工作。随着 WiFi 协议系列（802.11a/b/g/n）和 WiMAX 协议（802.16d/e/m）的推出，IEEE 构建了正交频分复用（OFDM）这一富有前景的先进空中接口技术。如 IEEE 802.11a WLAN 标准适用 5GHz 频带传送 OFDM 信号可实现 54Mbit/s 速率。在 2006 年，IEEE 标准化了新的 WiMAX（IEEE802.16m），引入基于组接入的无线基带系统。WiMAX 的特点包括 20MHz 可变带宽，更高的峰值数据传输率以及比当时的 UMTS 和 HSPA 更优秀的效率。这些强大优势迫使 3GPP 去推出可与 WiMAX 技术旗鼓相当的新无线移动标准。这一系列工作最终唤来了 LTE 协议的标准化。

1.2 数据速率的历史

表 1.1 总结了各个移动通信技术的峰值数据传输率。我们可以看到在这些标准支持的最大数据传输率中，LTE 标准（3GPP 标准第 8 发布版）提供了高达 300Mbit/s 的最大速率，而 LTE - Advanced（3GPP 标准第 10 发布版）更是可达到 1Gbit/s。

表 1.1 过去 20 年间推出的各个无线协议的峰值通信速率

技术	理论峰值数据速率（在低移动率状态下）
GSM	9.6kbit/s
IS - 95	14.4kbit/s
GPRS	171.2kbit/s
EDGE	473kbit/s
CDMA - 2000（1xRTT）	307kbit/s
WCDMA（UMTS）	1.92Mbit/s
HSDPA（Rel 5）	14Mbit/s
CDMA - 2000（1x - EV - DO）	3.1Mbit/s
HSPA +（Rel 6）	84Mbit/s
WiMAX（802.16e）	26Mbit/s
LTE（Rel 8）	300Mbit/s
WiMAX（802.16m）	303Mbit/s
LTE - Advanced（Rel 10）	1Gbit/s

从这些数据我们可以看出，数据速率已经比 GSM/EDGE 时代提升了 2000 多倍，比 W - CDMA/UMTS 时代提升了 50 ~ 500 倍。这个惊人的提升背后是近十多年新技术的不断发展和应用。我们可以毫不夸张地说，这些非同寻常的进步深深根植于 LTE 标准关键技术背后那些优雅的数学公式。去解释阐明这些关键技术和洞悉它们是如何协作达到如此出色的性能，这就是本书的目标。我们将会深入了解更多关于 LTE 标准 PHY（Physical Layer，物理层）的技术，以及如何仿真，验证和实现。

1.3 IMT - Advanced 要求

ITU 发布了一系列移动系统设计要求。第一个建议发布于 1997 年，称为 IMT - 2000（国际移动电信 2000）[1]。这个建议包括一系列针对无线接口规范的

目标和要求。3G 移动通信系统开发要求符合这些建议。随着 3G 系统的进化，IMT - 2000 要求也在过去十年间不断进化发展[2]。

在 2007 年，ITU 发布了一系列新的建议，为 IMT - Advanced 系统设定了更高的技术门槛和要求[3]。IMT - Advanced 的这些要求致力于构建真正的全球基带移动通信系统。这样的系统可以提供接入大范围基于组接入的先进移动通信服务，支持从低到高移动性的应用和宽范围数据速率，以及提供针对高质量多媒体应用的容量。ITU 发布这些新要求以刺激研究和开发活力，以期未来出现性能显著提升和超越 3G 系统的质量服务。

IMT - Advanced 一个标志性的特性，是针对高级服务与应用峰值强化提升数据速率（高移动率 100Mbit/s；低移动率 1Gbit/s）。这些要求为研究提供方向。3GPP 开发的 LTE - Advanced 标准和 IEEE 的移动 WiMAX 标准就是满足 IMT - Advanced 规范要求的两个代表。在本书中，我们将关注 LTE 标准，讨论它的物理层规范是如何涵盖满足了 IMT - Advanced 要求的。

1.4 3GPP 和 LTE 标准化

LTE 和 LTE - Advanced 标准由 3GPP 开发。它们继承了众多 3GPP 早期开发的标准（UMTS 和 HSPA）并在某种意义上说，它反映了这些技术的进步。不过，为了满足 IMT - Advanced 要求并保持和 WiMAX 标准的竞争力，LTE 标准需要从早期标准中采用的 W - CDMA 传输技术上大踏步飞跃。LTE 标准化工作开始于 2004 年并最终导致了大规模有野心的移动网络架构重建。在其后的 4 年里，在全球通信公司和网络标准化团体的密切关注下，LTE 标准化工作最终与 2008 年完成（3GPP 发布版本 8）。R8 版 LTE 标准晚些时候演进为 R9 版，反映了一些技术更新并随后更新为 R10 版，也就是我们所知的 LTE - Advanced 标准。LTE - Advanced 的进步表现在谱效率、峰值数据速率以及用户体验相关方面。随着峰值数据速率达到 1Gbit/s，LTE - Advanced 被 ITU 认可并作为 IMT - Advanced 技术。

1.5 LTE 要求

LTE 的各个要求涵盖了演进的 UMTS 系统架构的两个基本组件：UMTS 陆基接入演进（E - UTRAN），核心分组网演进（EPC）。整个系统的目标包括：

1）系统容量和覆盖扩展；

2）高峰值数据传输率；

3）低延迟（用户平台和控制平台）；

4）成本节约；

5）多天线支持；

6）可变带宽操作；

7）无缝兼容现有设备（UMTS，WiFi 等）。

作为 LTE 标准的主要任务，移动数据通信速率的实质性提升常常作为衡量通信研究，以及与移动通信链路最大可实现速率有关理论构思的标准。我们现在将会展示一些与此主题相关的精彩内容，这些灵感来自参考文献［4］中一个精彩的讨论。

1.6 理论策略

Shannon 关于信道容量状态的基础工作告诉我们数据速率永远受限于可接收到的信号功率（或接收信号信噪功率比）[5]。传输速率也与带宽相关。对于低带宽利用率来说，速率受到可利用带宽的显著制约，增大传输速率的努力都需成比例的增大接收信号功率。对于高带宽利用率来说，传输速率等于或大于可用带宽，所以比起成比例的增大带宽，相应的增大接收信号功率对传输速率有更多影响。

在接收端使用多路天线是一个比增加整体功率更好的办法。这就是我们说的接收分集。多路天线也可以在发送端使用，这就是我们说的发送分集。发送分集的方法基于波束形成，使用多路发送天线聚焦发送功率直接指向接收端。这个技术可以成指数增长接收信号功率从而达到更高数据传输速率。

不过，应用发送分集或接收分集以提高数据速率只能在特定的点有效。超过这个点，增加数据速率会造成饱和。一个替代方案是同时应用多路天线在发送端和接收端。比如，我们熟知的空域复用技术，其中使用多路天线在发送端和接收端。这是一类重要的多路天线技术，就是我们说的 MIMO。LTE 标准中包括不同类型的 MIMO 技术，包括开环和闭环空域复用。

除了接收信号强度之外，另一个参数也直接影响无线通信系统可达到的数据速率。它就是传输带宽。更高传输速率通常需要提供更宽的带宽。一个与宽带传送有关的重要的设计挑战就是无线信道的多径衰落效应。多径衰落是一个传输信号在到达接收端之前由于通过不同路径造成不同版本传播的现象。这些不同版本的信号叠加导致变化的信号功率特点和时延或相移。其结果是，接收端收到的信号会被调制成一个被信道脉冲响应滤波的样子。在频域上看，多径衰减信道表现为随时间变化的信道频率响应。信道频率响应不可避免地混杂在原始信号的频率特性中，从而对数据速率产生消极影响。为了调整信道频率选择性和得到满意的性能，我们必须在增加传输功率的同时，降低我们对数据速率的预期，或者用信

道平衡补偿频域特性的畸变。

很多信道平衡技术都可以克服多径衰落效应。采样时域平衡方法可满足一个5MHz 带宽下足够的传输性能。不过，对于拥有 10MHz、15MHz、20MHz 或更宽的带宽的 LTE 标准和其他移动系统，时域平衡器的复杂程度变得惊人复杂。为了克服时域平衡的问题，我们有两个办法可以适用于宽带传输中：

1）使用多载波传输方案：宽带信号可以处理成多路窄带正交信号的和。应用于 LTE 标准中多载波传输的例子就是 OFDM 传输。

2）使用单载波传输方案：OFDM 可以提供一个简单得多的频域平衡方法，不需要要求高传输功率起落。一个在 LTE 标准中的例子被称为单载波频分复用，应用于上行链路传输。

不仅如此，对在给定带宽实现更高数据速率有更大作用的办法是使用高阶调制。使用高阶调制允许我们在单调制符上搭载更多比特，提高带宽利用率。当然，高的带宽利用率也伴随着代价：减小了调制符之间的最小距离，因而对噪声和干扰变得更敏感。因此，在低或高阶调制中使用自适应调制和编码以及其他链路自适应策略是明智的办法。通过使用自适应方法，我们可以本质上提升通信链路的通过率和可实现的数据速率。

1.7 LTE 关键技术

LTE 以及其演进标准中诸关键技术包括了 OFDM，MIMO，turbo 编码和动态链路自适应技术。如我们前面讨论的这些技术，它们从起源到研究成熟应用于通信领域，最终结合到一起使 LTE 标准满足要求。

1.7.1 OFDM

如在第 6 发布版[6] 中优雅的描述，LTE 选择 OFDM 以及其单载波版本的 SC－FDM作为基本传输方案内容如下：针对多径衰减的可靠性高、频谱效率高、配置复杂度低、可支持可变传输带宽和如频率选择分配这样的高级功能、MIMO 传输和干扰协调。

OFDM 是一种多载波传输方案。它的主要思想产生晚于在多窄带正交副载波信道中排列数据符号的副载波调制，和在宽带信道频域内分割信息传送的技术。当在两个子载波频率间隔足够小的时候，OFDM 传送方案可以表现为一个频率选择性衰落信道，如同多个窄带平坦衰落子信道的集合。这使 OFDM 能够成为一个直观和简单的途径，通过发送已知数据和参考信号评估传输信道频率响应。凭借良好的接收端信道响应估计，我们随后可以使用低复杂度频域平衡器，复原发送信号的最好估计值。这个平衡器在某种意义上说转化了每个子载波的信道频率

响应。

1.7.2 SC - FDM

OFDM 多载波传输的一个缺点是瞬态发送功率的大范围波动。这对功率放大器产生消极影响而使移动终端基站负担较高的功率消耗。在上行两路传输中，设计一个复杂的功率放大器特别具有挑战性。因而，OFDM 传输的一个变体，就是我们所知的 SC - FDM 被引入 LTE 标准使用在上行链路传送。SC - FDM 一般和标准 OFDM 系统集成配置，并使用离散傅里叶变换（DFT）预编码[6]。通过使用 DFT 预编码，SC - FDM 根本上减小了发送功率波动的不利影响。因此，上行链路传输架构可以得到 OFDM 带来的更多好处，如低复杂度的频域平衡器和频域分配，而不需要对功率放大器设计提出那么硬性的要求。

1.7.3 MIMO

MIMO 是一个 LTE 标准中的核心技术。它深深根植于移动通信研究，支撑了 LTE 标准使用多径天线的优势，从而满足了峰值数据速率和通过率的要求。

MIMO 方法对移动通信的贡献体现在两个不同方面：提高总数据速率和提升通信链路的可靠性。应用于 LTE 标准的 MIMO 算法可分为 4 类：接受分集、发送分集，波束形成和空域复用。发送分集和波束形成算法，使我们可以在不同天线上发送随机信息。这两个方法不专注于提高可实现数据速率，而是提供通信链路更多的可靠性。空域复用有所不同，它使系统在不同的天线上发送独立（不随机）的信息。这个类型的 MIMO 方案可以在已有链路根本上提升数据速率。数据速率的提升程度可与发送天线数量线性成正比。LTE 标准在下行链路协议中提供了 4 径天线多路传送配置，已实现 MIMO。LTE - Advance 在下行链路上则允许使用最多 8 径发送天线。

1.7.4 Turbo 信道编码

Turbo 编码进化自卷积码。卷积码应用于所有的传统通信标准并贡献了出色的邻信道容量性能[7]。Turbo 编码出现于 1993 年，并被引入 3G UMTS 和 HSPA 系统。不过，在这些标准里，Turbo 编码只作为了提升系统性能的可选择方案。在 LTE 标准中，与以往不同，Turbo 编码作为了信道编码机制的唯一方案，用于处理用户数据。

Turbo 编码近乎理想的性能被 LTE 采用源自他的复杂度可计算和执行。LTE 中的 Turbo 编码为了有效的执行进行了很多改进。比如，通过附加 CRC（循环冗余检查）以检查 Turbo 编码器的输入，LTE Turbo 解码器可以在编码质量可以接受的情况下实现早期终止机制。这样，不用反复检查追踪解码的整个过程，解码

器就能在 CRC 检查无误的情况下早一些停止。这一简单的解决方案带来了 LTE Turbo 解码器有可计算复杂度减小，避免了不少性能上的损失。

1.7.5 链路自适应

链路自适应的定义是：一种可以调整和适应移动通信系统传输参数以更好响应通信信道的动态性的技术。根据信道质量不同，我们可以使用不同的调制和编码技术（适应性调制和编码），调整几个发送或接受天线（适应性 MIMO），甚至调整传输带宽（适应性波长）。在移动通信系统中和链路自适应关系很近的是信道依赖性规划。规划解决一个如何在两个不同用户间共享无线电资源的问题，以便更有效地利用这些资源。通常，我们既需要减小资源数量分配给每个用户，又需要按用户数据的类型和优先级匹配这些资源。当一个稳定信道条件下最好的服务质量要求可以被满足的时候，信道依赖性规划力求为足够多的用户提供资源。

1.8 LTE 物理层建模

在本书中，我们将关注无线电接入网络物理层的数字信号处理。我们基本上在这里不讨论 LTE 核心网络，也不涉及更高层的处理比如像无线电资源控制（RRC），无线电链路控制（RLC）以及媒体接入控制（MAC）这些技术。

物理层建模包括所有数据比特从更高层传输到物理层的处理。它表现为一些列传输信道如何映射到物理信道，信号处理如何在每一个物理信道工作，以及数据如何最终被传送到天线而被发送。

如图 1.2 所示，这是一个 LTE 下行链路传输的物理模型。首先，数据被阶梯化多路编码，也就是我们说的下行链路信道共享处理（DLSCH）。DLSCH 处理包括附加 CRC 编码用于检差误码，对用户数据进行 Turbo 编码，进行比特率匹配操作选择几个输出比特去表示编码率，以及最后把码组重组和为码字。下一步就是所谓下行物理链路信道共享处理（PDSCH）。在这一步，对码字首先进行绕码操作，然后进行调制映射以形成调制过的符号流。再下一步包括 LTE MIMO 或多径天线处理，调制过的符号信号流被分为指定的多路子信号流通过多径天线传输。MIMO 操作可以表示为另个步骤：预编码和层映射。预编码组织符号分配到每一个子数据流，层映射选择和路由数据到子数据流，配置成 MIMO 下行链路传输中 9 种不同模式中的一个。所有 MIMO 在下行链路中都以传输分集，空域服用和波束形成实现。最后工作是和多载波传输相关的一些列处理。在下行链路中，多载波操作基于 OFDM 传输方案。OFDM 传输也包括两个步骤：首先，使用时域－频域资源网格，把资源元素映射到每一层的调制符号上。在调制网格的频轴

上，数据和子载波的频域相关。在 OFDM 信号生成阶段，应用反傅里叶变换生成一系列的 OFDM 符号，以及时计算发送数据并传输到每个天线。

在作者看来，这一系列强大和富有创意的传输结构可以将所有关键技术如此有效率的结合以符合 IMT - Advanced 各种严格的要求，并最终形成 LTE 标准。通过关注 PHY 建模，我们设置挑战以期结合 LTE 标准理解数据信号处理的发展。

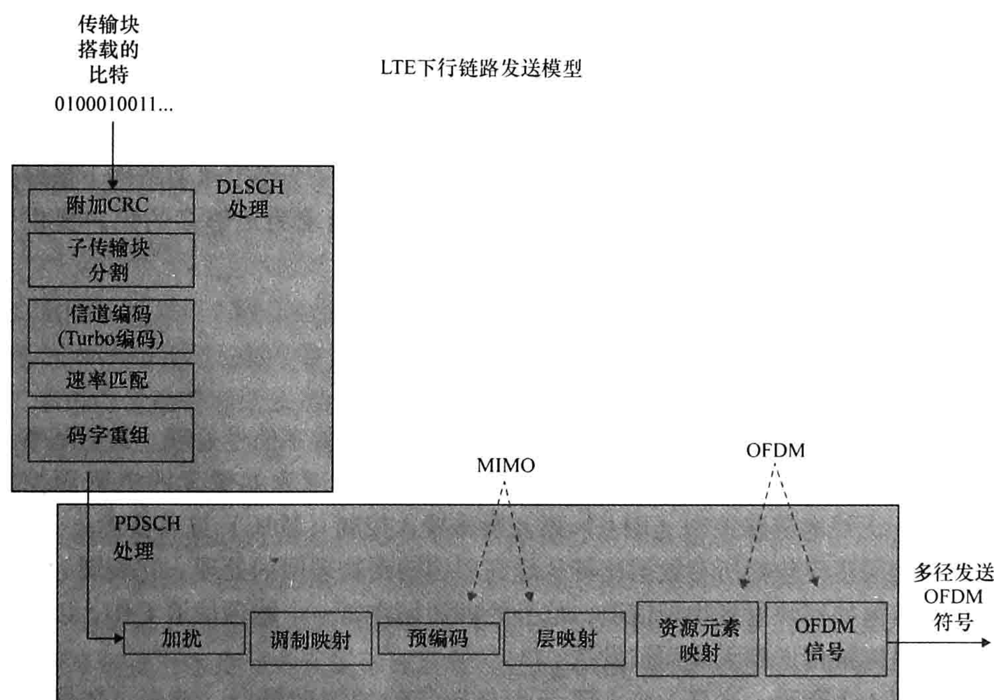


图 1.2　LTE 标准中的物理层协议

1.9　LTE（R8 版和 R9 版）

LTE 的第一个版本的发布，是3GPP 将从2005 年起近4 年工作的顶点。随着对各种技术能力是否满足 LTE 的要求的一系列扩展性研究，3GPP 决定新的空中接口传输技术标准将基于下行链路 OFDM 和上行链路 SC - FDM。全部规范，包括多种 MIMO 模式，随后被吸收进协议。LTE 标准的第一个版本（3GPP，R8 版）发布于2008 年 12 月。R9 版发布于 2009 年 12 月；它的内容相应地反映了一些最新技术进展如支持多媒体广播/组播服务（MBMS），定位服务和提供基站支持多种协议。

1.10 LTE - Advanced（R10 版）

LTE - Advanced 发布于 2010 年 12 月。LTE - Advanced 是原始 LTE 标准的演进，并不表示一个最新技术的出现。一些列技术被 LTE 吸收最终使 LTE - Advanced 成为包括载波聚合，增强型下行链路 MIMO/上行链路 MIMO 和分程传递的标准升级版。

1.11 MATLAB 和无线系统设计

在本书中，我们使用 MATLAB 对 LTE 标准的 PHY 层进行建模，洞察仿真和实现的要求。MATLAB 是一个广泛应用于数学建模和数值计算的编程和高级开发环境。我们同时使用 Simulink，一个针对系统仿真和模型设计的图形化环境，以及一系列 MATLAB 工具箱——组件的面向应用库以方面测试使用 MATLAB 建模的应用。比如，对通信系统建模，我们使用通信系统工具箱的功能。这个工具箱包括了 MATLAB 和 Simulink 里的无线协议，提供设计，原型生成，仿真和通信系统验证的工具。

系统对象，是本书中介绍的 MATLAB 的各种功能中一个新的功能。系统对象是可在 MATLAB 很多工具箱中使用的一系列构建模块的算法，以适合系统设计。它们是一些自我归档的算法，在 MATLAB 测试平台中方便测试系统仿真。通过覆盖大部分算法，系统对象也可以满足我们使用 MATLAB、C 或其他语言重新构建通信系统基本模块的需求。系统对象不仅可以建模和仿真，它也可以提供执行功能。比如，它具有的突出特性包括帮助仿真提速、支持 C/C + + 代码生成、定点数表示，并难得地支持 HDL（Hardware Description Language，硬件描述语言）代码自动生成。

1.12 本书组织结构

本书的标题是通过理解四个关键技术（OFDMA、MIMO、Turbo 编码和链路自适应），读者可以深入理解 LTE 标准的物理层模型。第 2 章涉及对 LTE 标准各技术规范的一个简短概览。第 3 章介绍在 MATLAB 建模和仿真移动通信系统中会使用到的的工具和功能。第 4 ~7 章，我们逐一详细讲解 OFDM、MIMO、调制，编码和链路自适应技术。在每章中，我们使用 MATLAB 建模一步步创建并反复使用基于这些关键技术的 LTE 物理层组件。在第 8 章，在系统级规范和性能评价中，我们讨论各种标准中涉及的信道模型，使用 MATLAB 和 Simulink 定量和定性的分析系

统级的性能。这章里，作为这本书第一部分的结束，我们会通过集合一个系统模型以展示 LTE 的 PHY 如何在 MATLAB 里建模对前面各章节的深入工作进行小结。

本书的第二部分将深入讲解一些诸如系统仿真和组件执行的实际问题。第 9 章讨论如何通过一些方法提升 MATLAB 程序速度，如并行计算、C 代码自动生成，GPU（图像处理单元）参与运算，以及使用更高效的算法。在第 10 章里，我们讨论与执行有关的一些问题，比如从 MATLAB 代码如何自动生成 C/C + + 代码，目标环境、代码优化以及编程风格的影响。作为硬件实现的先决条件，我们也会讨论数据的定点数表示法，以及它如何影响一系列模型组件。最后，在第 11 章，我们总结前面的讨论并对未来的工作进行一个前瞻性的展望。

参 考 文 献

[1] ITU-R (1997) International Mobile Telecommunications-2000 (IMT-2000). Recommendation ITU-R M. 687-2, February 1997.
[2] ITU-R (2010) Detailed specifications of the radio interfaces of international mobile telecommunications-2000 (IMT-2000). Recommendation ITU-R M.1457-9, May 2010.
[3] ITU-R (2007) Principles for the Process of Development of IMT-Advanced. Resolution ITU-R 57, October 2007.
[4] Dahlman, E., Parkvall, S. and Sköld, J. (2011) *4G LTE/LTE-Advanced for Mobile Broadband*, Elsevier.
[5] Shannon, C.E. (1948) A mathematical theory of communication. *Bell System Technical Journal*, 379–423, 623–656.
[6] Ghosh, A. and Ratasuk, R. (2011) *Essentials of LTE and LTE-A*, Cambridge University Press, Cambridge.
[7] Proakis, J.G. (2001) *Digital Communications*, McGraw-Hill, New York.

2 LTE 物理层概览

本书关注 LTE（长期演进计划）无线电接入技术，特别是其物理（PHY）层。在这里，我们将会重点针对设计 LTE 物理层无线电接口技术的主要思想。关注这些要点可以更好地解释 LTE 和 LTE - Advanced 标准如何达到数据速率的目标。

LTE 为上行链路（移动端到基站）和下行链路（基站到移动端）定义了数据通信协议。在 3GPP 命名习惯上，基站一般命名为 eNodeB（enhanced Node Base station，增强型节点基站），移动单元一般命名为 UE（User Equipment，用户设备）。

在本章中，将总括 LTE 标准中 PHY 数据通信和传输协议的要点，我们将首先概览 LTE 标准的频带、FDD（Frequency Division Duplex，频分双工）和 TDD（Time Division Duplex，时分双工）这些双工方法论，以及可变带宽调度、时间帧和时 - 频资源分布。我们随后详细研究上行链路和下行链路处理堆栈，它包括多载波传输方案、多径天线协议、自适应调制，和编码方案以及信道相关性链路自适应原理。

在每个部分，我们会首先描述连接通信堆栈不同层之间的各种信道，随后详细描述下行链路和上行链路 PHY 层的信号处理过程。这一系列详细阐述，将为我们使用 MATLAB 建立下行链路 PHY 层模型提供足够的技术准备。在随后的四章中，我们将会反复逐步从 MATLAB 比较简单的算法中构建系统模型。

2.1 空中接口

LTE 的空中接口在下行链路基于 OFDM（正交频分复用）多路接入技术，在上行链路基于类似的技术：SC - FDM（单载波频分复用）。OFDM 具有可变多路接入的众多优势，并超越了以往技术。这些优势有频带效率高和对基带数据传输的适应性高，对多径衰落带来的码间干扰的高抗性。它天生支持 MIMO 方案和多种频域技术，如频率选择性调度[1]。

OFDM 的时 - 频分布在传输中同时分配频谱和时间帧方面提供了高度的灵活性。LTE 的频谱灵活性不仅表现在频带多样化上，更表现在对带宽可变的设置。LTE 同时支持 10ms 的短帧来减少延迟。通过定义短帧大小，LTE 可以更好地评估移动端的信道，实时地为基站的链路自适应提供必要的反馈。

2.2 频带

LTE 标准定义了在不同频带上可用的无线电频率位置。LTE 标准的一个目标是无缝兼容旧移动系统。因此，过去 3GPP 标准中已定义使用的频率在 LTE 开发中依然可以使用。除了这些通用频带，LTE 协议也第一次引入了几个新频带。这些新频带随不同国家间管制规定不同而不同。因此可以想象，不仅仅是一个而是很多频带都可被指定服务商使用，从而构建更方便运营的国际漫游服务机制。

在 3GPP 其他早期协议中定义了 FDD 和 TDD 模式，故 LTE 也对它们支持，并在频带上分别定义为成对频谱和非成对频谱。FDD 频带为成对频谱，它可以同时在两个频率上传输：下行链路使用一个，上行链路使用一个。成对频带有足够的间隔以提升接收器性能。TDD 频带是非成对频谱，上行链路和下行链路传输共用同一信道和载波频率。这中传输在上行链路和下行链路上是时间复用的。

3GPP LTE 的协议第 11 发布版内有一个简单易懂的 ITU IMT - Advanced 频带表[2]。它包括 FDD 的 25 个频带和 TDD 的 11 个频带。如表 2.1 所示，FDD 双工模式的成对频带从 1 到 15 编号；TDD 模式的非成对频带从 33 到 43 编号。频带 6 不在 LTE 中使用。15 和 16 频带为 ITU 区域 1 预留，见表 2.1 和表 2.2。

表 2.1 E - UTRA 规定的成对频带

工作频带编号	上行链路（UL） 工作频带频率范围/MHz	下行链路（DL） 工作频带频率范围/MHz	双工模式
1	1920 ~ 1980	2110 ~ 2170	FDD
2	1850 ~ 1910	1930 ~ 1990	FDD
3	1710 ~ 1785	1805 ~ 1880	FDD
4	1710 ~ 1755	2110 ~ 2155	FDD
5	824 ~ 849	869 ~ 894	FDD
6	830 ~ 840	875 ~ 885	FDD
7	2500 ~ 2570	2620 ~ 2690	FDD
8	880 ~ 915	925 ~ 960	FDD
9	1749.9 ~ 1784.9	1844.9 ~ 1879.9	FDD
10	1710 ~ 1770	2110 ~ 2170	FDD
11	1427.9 ~ 1447.9	1475.9 ~ 1495.9	FDD
12	699 ~ 716	729 ~ 746	FDD
13	777 ~ 787	746 ~ 756	FDD

（续）

工作频带编号	上行链路（UL） 工作频带频率范围/MHz	下行链路（DL） 工作频带频率范围/MHz	双工模式
14	788～798	758～768	FDD
15	预留	预留	FDD
16	预留	预留	FDD
17	704～716	734～746	FDD
18	815～830	860～875	FDD
19	830～845	875～890	FDD
20	832～862	791～821	FDD
21	1447.9～1462.9	1495.9～1510.9	FDD
22	3410～3490	3510～3590	FDD
23	2000～2020	2180～2200	FDD
24	1626.5～1660.5	1525～1559	FDD
25	1850～1915	1930～1995	FDD

表 2.2 E－UTRA 规定的非成对频带

工作频带编号	上行链路和下行链路 工作频带频率范围/MHz	双工模式
33	1900～1920	TDD
34	2010～2025	TDD
35	1850～1910	TDD
36	1930～1990	TDD
37	1910～1930	TDD
38	2570～2620	TDD
39	1880～1920	TDD
40	2300～2400	TDD
41	2496～2690	TDD
42	3400～3600	TDD
43	3600～3800	TDD

2.3 单播和组播服务

对移动通信来说，一般传输模式是我们熟知的单播传输。它只对单一用户传

输数据。除了单播模式之外，LTE 支持多媒体广播/组播服务（MBMS）的传输模式。MBMS 提供如 TV 和广播以及音频视频流这样高数据速率的多媒体服务[1]。

MBMS 拥有一套它自己的专有业务和控制信道，以及基于多校区传输方案而形成的多媒体广播单频网络（MBSFN）。多媒体信号从属于一个指定的 MBSFN 服务区内多个相邻的小区发送。当一个多播信道内容从不同小区发送，在相同子载波的信号会在 UE 连贯组合起来。这使得 SNR（Signal - to - Noise Ratio，信噪比）和多媒体传输的最大上限数据速率得到本质提升。不管是单播还是组播传输都会影响众多系统运行组件和参数。对我们提及的 LTE 技术各个组件，我们将会着重突出不同的信道、传输模式以及物理信号和参数是如何在单播和组播模式运行中工作的。本书的重点将集中在单播服务和数据传输。

2.4　带宽分配

IMT - Advanced 对 LTE 标准的频谱灵活性提出了指导要求。这表现在频域的可伸缩性，概括在一个 1.4 ~20MHz 频谱范围的列表中。LTE 的频率范围和包括 12 个子载波的资源块互相关联。这些子载波以 15kHz 等分，所以资源块的总带宽是 180kHz。传输带宽可以在单一频率的载波上配置 6 到 110 个资源块，这样 LTE 标准的多径传输特性就提供了 1.4 ~20MHz 每 180kHz 递增的信道带宽，并保证了所要求的频谱灵活性，如图 2.1 所示。

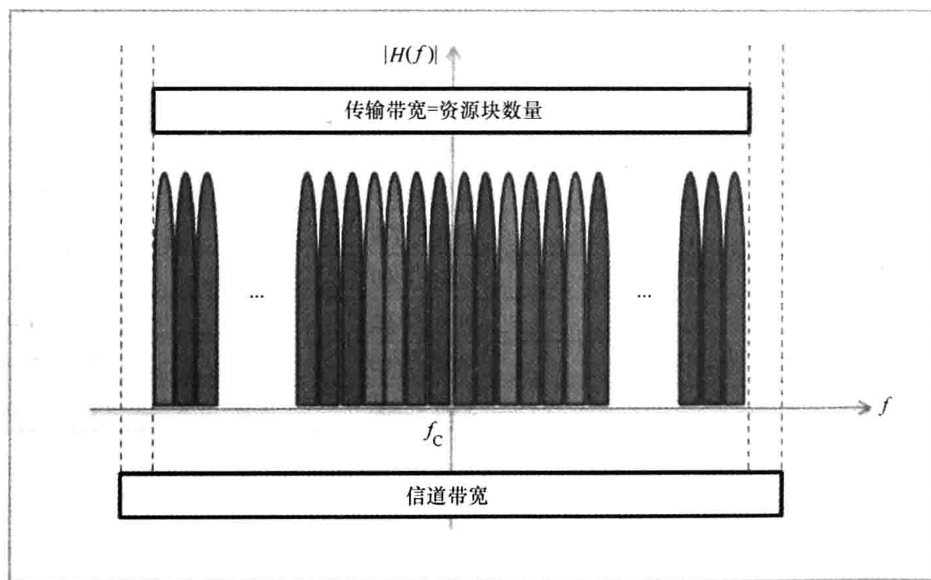


图 2.1　信道带宽与资源块数量的关系

表 2.3 展示了覆盖 LTE RF 载波资源块数量和信道带宽之间的关系。在带宽 3~20MHz 范围内，传输带宽内的资源块占据了 90% 的信道带宽。在 1.4MHz，这一比例降到 77% 左右。这将有助于降低不希望出现的带外发射（见图 2.1）。频谱的时－频分布，以及资源网格和区块的标准定义将会在稍后给出。

表 2.3 LTE 信道带宽定义

信道带宽/MHz	资源块数量
1.4	6
3	15
5	25
10	50
15	75
20	100

2.5 时间帧

图 2.2 所示为 LTE 的时域结构。深入理解 LTE 传输过程取决于清晰理解数据的时－频分布，它如何映射到资源网格，以及资源网格是如何最终转变成 OFDM 符号而传输的。

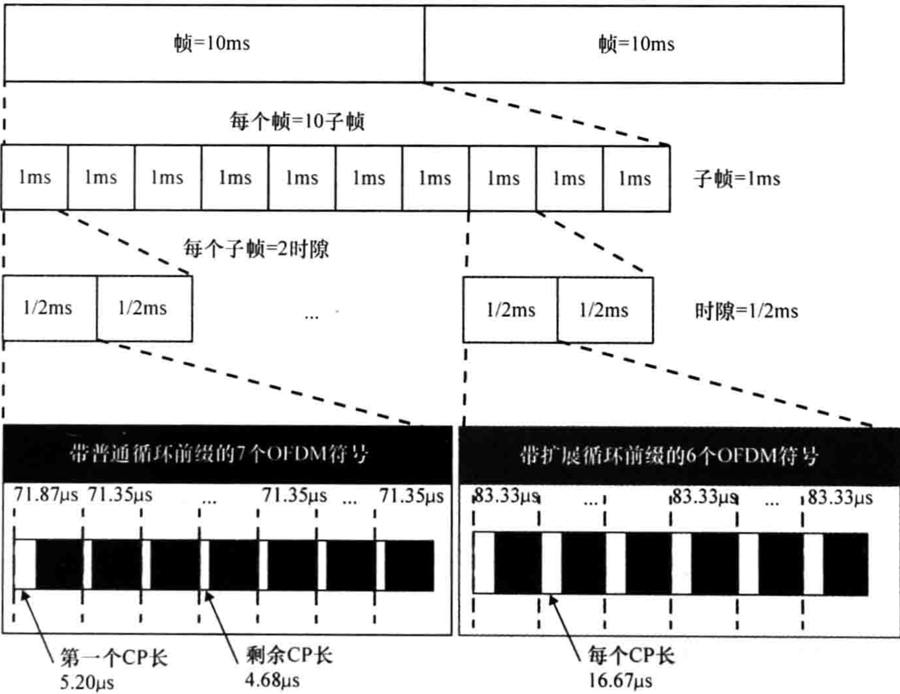


图 2.2 LTE 时域结构

在时域，LTE 以 10ms 长度的无线电帧序列传输。每一个帧可以细分为 10 个长度为 1ms 的子帧。每一个子帧由两个长度 0.5ms 的时隙组成。每个时隙包含若干个 OFDM 符号。这些 OFDM 符号一般有 6 个或 7 个，取决于使用普通循环前缀或是使用扩展循环前缀。

2.6 时－频分布

OFDM 最吸引人的一个特点就是对发送信号明确映射了时－频分布。在编码与调制之后，复调制信号的变体——物理资源元素，被映射到时－频坐标系统——资源网格中。资源网格在 x 轴方向为时间，在 y 轴方向为频率。x 轴的资源元素是与时间相关的 OFDM 符号。y 轴方向表示与频率相关的 OFDM 子载波。

图 2.3 所示为使用普通循环前缀的 LTE 下行链路资源网格。一个 OFDM 符号和子载波方向的交叉点对应一个资源元素。子载波间隔 15kHz。在使用普通循环前缀的情况下，每一个子帧有 14 个 OFDM 符号（每一个时隙有 7 个符号）。资源块定义为在频域上 12 个载波或 180kHz，在时域上一个 0.5ms 时隙构成的资源元素组。当使用普通循环前缀的情况下，每一个时隙有 7 个 OFDM 符号，这样每一个资源块包括了 84 个资源元素。在使用扩展循环前缀的情况下，每一个时隙有 6 个 OFDM 符号，这样每一个资源块包括了 72 个资源元素。资源元素的定义非常重要，因为它是时域调度的传输最小单元。

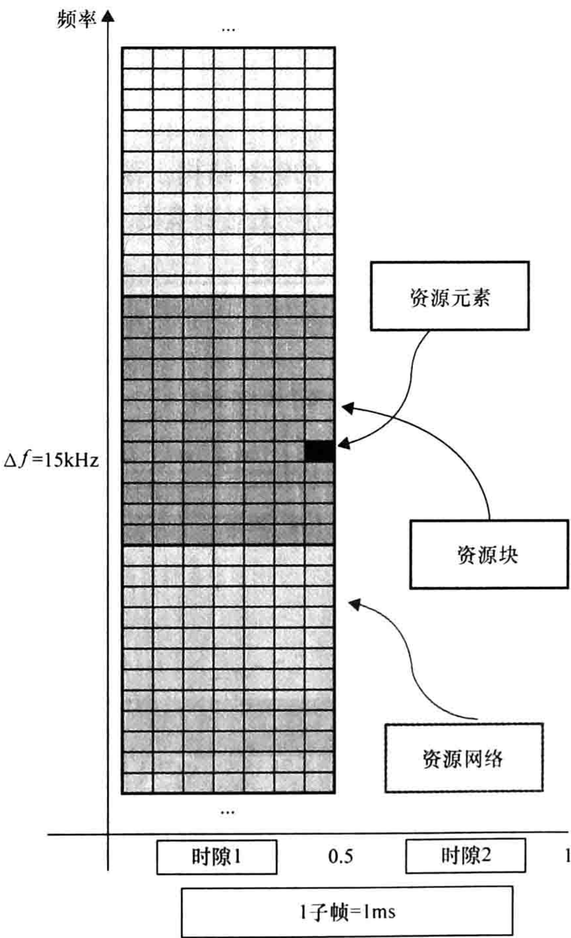


图 2.3　资源元素，块，和网格

在前面的讨论中，LTE

PHY 层允许 RF 载波在频域上包含若干个资源块。从最小 6 个到最大 110 个。它对应了传输带宽 1.4 ~ 20.0MHz 每 15kHz 递增的范围，支持了 LTE 高带宽灵活性。在上行链路和下行链路中都应用了资源块定义。下行和上行链路有一点点区别在于子载波的中心频率有所不同。

在上行链路中，没有不使用的 DC 频率成分的子载波，而上行链路载波中心频率定义在两个子载波中间，如图 2.4 所示。在下行链路，中心位置频率不使用，如图 2.5 所示。下行链路传输中不使用 DC 子载波是为了避免非比例高串扰出现的概率。

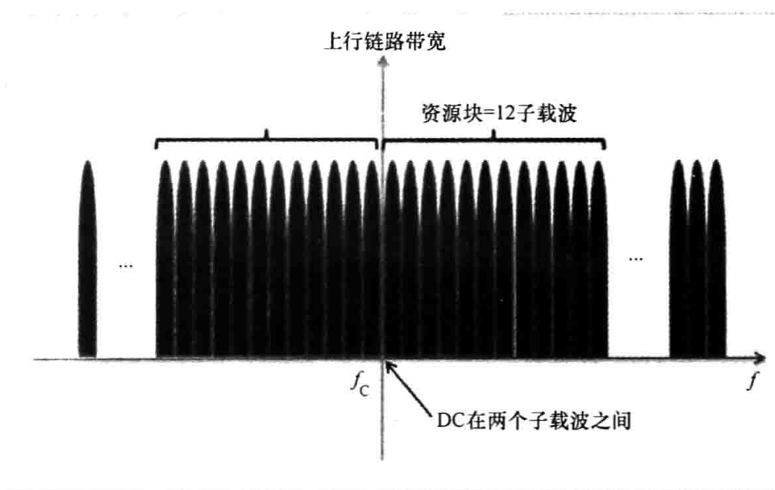


图 2.4　资源块和上行链路中 DC 频率成分

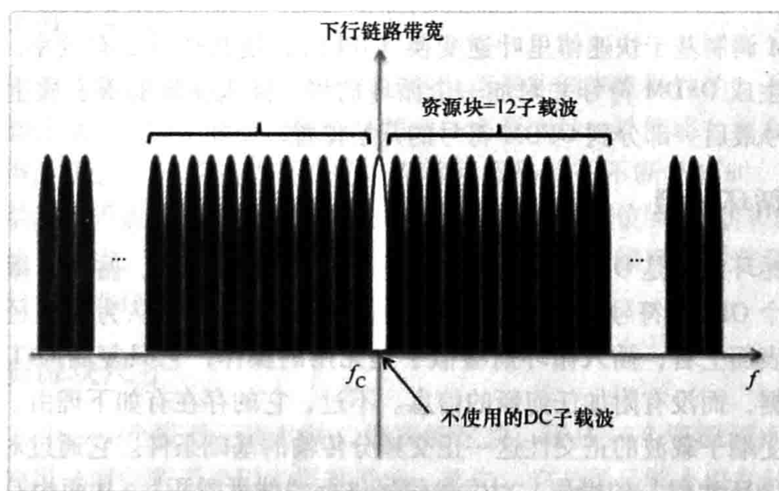


图 2.5　资源块和下行链路中 DC 频率成分

选择 15kHz 的频率间隔可以完美匹配 OFDM 转换频率选择性信道到恒定频率子信道的良好分辨率。此转换通过在每一个平坦衰落子信道使用一组低复杂度的平衡器，帮助 OFDM 在频域上更有效地降低频率选择性衰落。

2.7　OFDM 多载波传输

在 LTE 标准中，下行链路传输基于 OFDM 方案，而上行链路基于与 OFDM 类似的 SC－FDM 方案。OFDM 是一个多载波传输的方法，表现在它的基带传输带宽是若干个窄带子信道的集合。

OFDM 信号生成包括多个步骤。首先，调制数据被映射到由频域组织和分配的资源网格中。每个调制符号 a_k 对应频轴上的一个子载波。当 N 个子载波以 Δf 间隔占据带宽时，带宽和子载波间隔的关系为

$$BW = N_{rb}\Delta f \tag{2.1}$$

每个子载波 f_k 可以认为是多个子载波间隔的集合：

$$f_k = k\Delta f \tag{2.2}$$

OFDM 调制包括一组 N 个复调制器，每个调制器对应一个子载波。因而 OFDM 调制输出 $x(t)$ 可表示为

$$x(t) = \sum_{k=1}^{N} a_k e^{j2\pi f_k t} = \sum_{k=1}^{N} a_k e^{j2\pi k\Delta f t} \tag{2.3}$$

假设信道采样率是 F_s，信道采样时间为 $T_s = 1/F_s$，则 OFDM 调制的离散时域分布可表示为

$$x(n) = \sum_{k=1}^{N} a_k e^{j2\pi k\Delta f n/N} \tag{2.4}$$

OFDM 调制基于快速傅里叶逆变换（IFFT），使其执行更有效率。OFDM 调制之后，生成 OFDM 符号并附加一个循环前缀。插入循环前缀实质上就是复制 OFDM 符号最后一部分到 OFDM 符号的开始位置。

2.7.1　循环前缀

插入循环前缀是 OFDM 信号生成过程的一个重要功能。循环前缀可以有效防止前一个 OFDM 符号发送中带来的干扰。码内干扰可以认为是多径传播的直接影响。表面上看，插入循环前缀似乎是无用的操作，它只复制 OFDM 符号中存在的数据，而没有附加任何新的信息。不过，它的存在有如下理由。首先，它能确保接受端子载波的正交性这一正交频分传输的基础条件。它通过对信道进行近似于“循环卷积”的操作，对发送信号进行“线性卷积”，从而为 OFDM 信号提供周期性扩展。对于 OFDM 在频域表示调制信号，模仿循环卷积的循环前缀

是非常重要的。在接收端频域平衡的有效性只能通过信道响应表现为循环卷积时，也就是插入循环前缀才能保证[2]。

循环前缀的长度对多载波传输系统是一个重要的设计参数。一方面，循环前缀的长度必须足够长，以覆盖小区通信环境里大多数传播情形中通常的延时扩散。另一方面，循环前缀作为随机数据和必要的开销，就如“前缀”这个词的含义，接受到的 OFDM 信号第一部分在接收端都会被丢弃。因此，LTE 必须保证循环前缀尽可能小，以减小这些开销而最大化频谱效率。为了平衡这一矛盾，LTE 保证循环前缀长度为所期望的传播信道延时扩散，以及为时许调校不良造成的误码提供一定的裕度。

如表 2.4 所示，LTE 标准包括三个不同的循环前缀值：普通（4.7μs）、对间隔 15kHz 子载波的扩展（16.6μs）和对间隔 7.5kHz 子载波的扩展（33μs）。

表 2.4 普通和扩展循环前缀规定

配置	子载波间隔 Δf/kHz	每个资源块内子载波数量	每个资源块内 OFDM 符号数量
普通循环前缀	15	12	7
扩展循环前缀	15	12	6
	7.5	24	3

需要注意间隔 7.5kHz 子载波之在组播/广播时使用。4.7μs 的普通循环前缀长度可以匹配大多数城乡通信环境，反映这些环境里延时扩散的一般值。鉴于每个 OFDM 调制符号所占用的时间大约是 66.7μs，普通循环前缀开销其 7%。扩展模式下这一开销是 25%。对野外环境的传输来说，随着延时扩散增大以及广播服务情况不同，循环前缀的开销也有必要变得更大。

2.7.2 子载波间隔

很小的子载波间隔保证每一个子载波的衰落是非频率选择性的。不过，子载波间隔不能任意小。当子载波间隔减小超过一定限度后，性能将会因多普勒频移和相位噪声而变差[1]。多普勒频移发生在移动端移动并不断加速时。多普勒频移会导致载波间干扰，导致小载波间隔下的劣化放大。相位噪声或抖动是由于本振的频率波动造成，并会导致载波间干扰。为了减小相位噪声和多普勒频移造成的劣化，LTE 标准规定子载波间隔为 15kHz。

2.7.3 资源块尺寸

在 LTE 中，一个资源元素的块，即资源区块，形成一个资源调度单元。在选择资源块尺寸时，若干个因素需被考虑。首先，它必须足够小以在频率选择性调度中占优（如在良好频率子载波上调度数据传输）。小资源块尺寸保证每个资

源块的频率响应较小，而使调度器只分配那些良好的资源块。不过，eNodeB 不清楚哪个资源块处于好的信道条件下，这个信息须由 UE 反馈。因此，资源块必须足够大以避免过度的反馈开销。当 LTE 的子帧大小为 1ms 以保证延迟时，资源块尺寸在频率上应很小，这样可以有效支持小数据包。因此，LTE 选择 180kHz（12 子载波）作为资源块带宽。

2.7.4 频域调度

LTE 支持不同系统带宽。OFDM 和 SC – FDM 通过 IFFT 操作产生发送信号。我们因此可以通过选择不同的 FFT 长度而得到不同的带宽。忽略被使用的带宽，LTE 保持 OFDM 符号时长一定，为 66.7μs。这使相同 15kHz 子载波可用于所有带宽。这一设计选择保证了相同频域平衡技术可以应用跨越多个频带。固定的符号时长也意味着在不同频带上有相同的子帧长度，这一特性使传输模型中时间帧定义得到大大简化。即使在实际的 FFT 在每个带宽上长度并没有标准定义，20MHz 上 FFT 长度通常为 2048。其他频带上 FFT 长度也通常成比例缩小，见表 2.5。

表 2.5 资源块、FFT 和每个 LTE 带宽的循环前缀长度

下行链路传输子帧时长 1ms、子载波间隔 15kHz 的 OFDM 参数						
带宽/MHz	1.4	3	5	10	15	20
采样频率/MHz	1.92	3.84	7.68	15.36	23.04	30.72
FFT 长度	128	256	512	1024	1536	2048
每个资源块内	6	15	25	50	75	100
OFDM 符号数量	14/12				（普通循环前缀/扩展循环前缀）	
CP 长度	4.7/5.6				（普通循环前缀/扩展循环前缀）	

2.7.5 接收端典型操作

对于接收端的操作，我们以发射端的反向过程处理。尽管 LTE 标准如其他标准一样，没有规定接收端一侧的操作，但讨论接受端典型操作有助于我们理解标准定义的发射端侧操作背后的动机。

OFDM 接受端反转了 OFDM 信号生成和发送的过程。首先，我们从接收到的 OFDM 符号的开始删除循环前缀。随后，通过 FFT 操作，我们计算接收到的一个 OFDM 符号内的资源网格元素。在这一步，我们需要在接受到的资源元素上进行平衡操作，来消除信道和码内串扰的影响，以恢复发射资源元素的最好估计。

在资源元素上进行平衡，我们首先需要对所有带宽估计信道频率响应；这一过程针对所有资源元素。引入引导符或小区参考信号（CSR）的重要性是显而易

见的。通过在资源网格上多个已知点发送一个已知信号作为引导符，我们可以在相应子信道轻松估计实际的信道响应。这些信道响应可以通过各种方式计算，如通过一个接受信号和发送信号的简单比值。现在我们得到了一些资源网格上标准点的信道响应，接下来可以进行各种平均或插值操作来估计所有资源网格的信道响应。估计资源网格的信道响应之后，我们可以通过把信道响应估计值的倒数乘以接受到的资源资源元素，得到资源元素发送值的最好估计。

2.8 单载波频分复用

LTE 上行链路基于 OFDM 传输方案的一个变体，即 SC - FDM。SC - FDM 减小 OFDM 传输中出现的瞬态频率波动。因此，它对于设计满足用户端（UE）低功耗放大器是最好的选择。LTE 标准中 SC - FDM 实质上是通过一个带 DFT（离散傅里叶变换）预编码器的 OFDM 调制器执行的。这一技术即离散傅里叶变换扩展正交频分复用（DFTS - OFDM）。

与单载波传输的不同之处，在于每个数据符号实质上分散于所有所用带宽上。对比 OFDM，每个数据符号只分布在一个子载波上。通过在所有带宽上分散数据功率，SC - FDM 减少了传输功率的有效值并保证了传输信号在功率放大器线性区域内的动态范围。SC - FDM 同样拥有 OFDM 所有的优势，包括保证多个上行链路用户的正交性、用频域平衡恢复数据以及克服多径衰落。不过，SC - FDM 的性能对于同一接收端来说一般弱于 OFDM[1]。DFTS - OFDM 会在随后章节讨论。

2.9 资源网格的内容

LTE 传输方案依据 OFDM 循环前缀长度不同，每个 1ms 子帧有 12 或 14 个 OFDM 符号的时间分辨率。对于频率分辨率，根据带宽不同，它提供从 6 到 100 个资源块，每个资源块包含 12 个 15kHz 间隔的子载波。一个问题是，什么类型的数据占据资源网格中的资源元素。这个答案需要我们讨论构成资源网格内容的各种物理信道和信号。

三种类型的信息实际存在于物理资源网格。每个资源元素既包括用户数据的调制符号，也包括参考信号、同步信号和来自更高层信道的控制信息。图 2.6 所示为在单播模式下定义的用户数据、控制信息和参考信号在资源网格中的位置。

对单模模式，用户数据搭载了每个用户想要通信的信息，它们以传输块方式从 MAC（媒体介入控制）层发送到物理层。多种类型的参考和同步信号在基站和移动设备上生成。这些信号用于如信道估计、信道测量，同步这些用途。最后

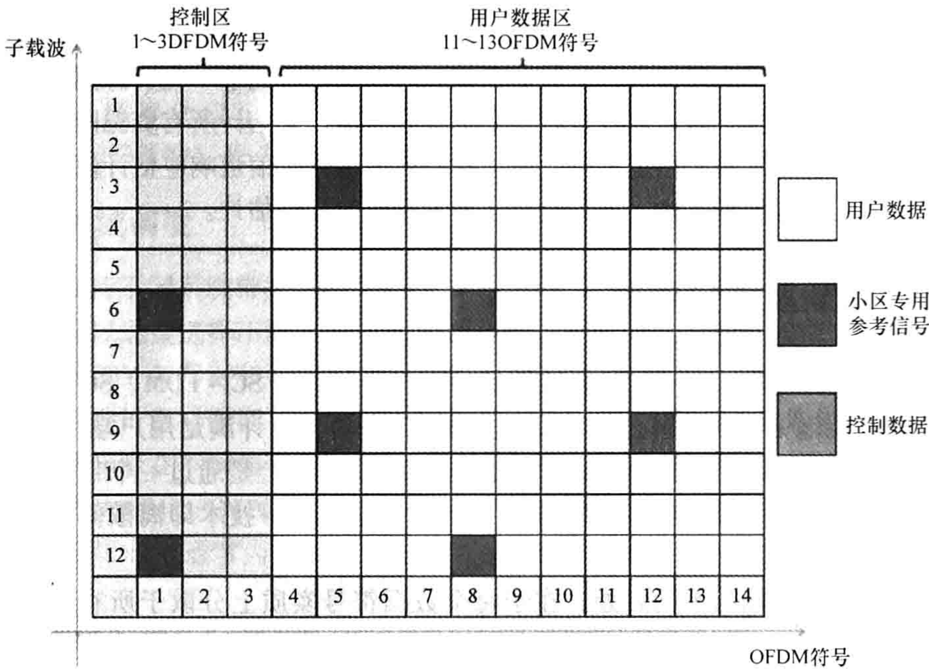


图2.6 单播模式下LTE下行链路子帧内的物理信道和信号内容

还有各种类型的控制信息，它们通过控制信道携带了接受端需要的用以正确解码信号所需的信息。

接下来，我们将描述上行链路和下行链路传输使用的物理信道，以及它们与上层信道之间的关系；即传输信道和逻辑信道。比较UMTS（通用移动通信系统）和其他3GPP标准，LTE极大程度减少了使用专用信道，取而代之更大程度的使用公共信道。这就解释了公共物理信道上集中了多种不同类型的逻辑信道和传输信道的特点。除了物理信道之外，两种类型的物理信号——参考信号和同步信号——也在公共信道上发送。LTE信道和信号将在下面的部分详细说明。

2.10 物理信道

纵观LTE标准的各个目标可归一为构造一个更有效和流线型的协议栈和架构。3GPP先前标准中定义的很多专用信道都被公共信道替代，物理信道总数也被削减。图2.7所示为无线电接入网络的协议栈和它的层结构。

逻辑信道体现了无线电链路控制（RLC）层和MAC层的数据传输和互联。LTE定义了两种逻辑信道：业务信道和控制信道。业务信道传输用户平面数据。

传输信道连接MAC层和物理层，物理信道在物理层上由收发端实现。每个

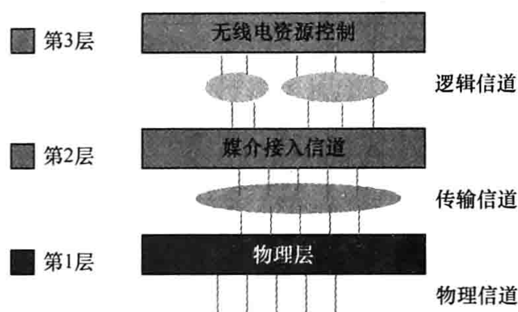


图 2.7 LTE 无线电接入网络的层架构

物理信道由一组资源元素构成，这些资源元素搭载了用于空中接口上最终传输的上层信道协议栈。在下层或上层链路数据传输分别使用 DL - SCH（下行链路公共信道）和 UL - SCH（上行链路公共信道）这两种传输信道。一个物理信道搭载特定传输信道传输使用的时 - 频资源。每个传输信道映射到相应的物理信道。除此以外，也有物理信道和传输信道没有一一映射的情况。这些信道，即 L1/L2 控制信道，用于下行链路控制信息（DCI），它提供适时接收和下行链路数据解码的信息终端，以及上行链路控制信息（UCI），提供调度器和携带终端状况信息的混合自动重传请求协议。LTE 中逻辑信道、传输信道和物理信道在上行链路和下行链路中不同。下面我们将会描述各种在下行链路和上行链路中的物理信道，以及它们与上层信道的关系和它们搭载的信息。

2.10.1 下行链路物理信道

表 2.6 总结了 LTE 下行链路物理信道。物理组播信道（OMCH）用于 MBMS。其余物理信道用于传统的单播模式传输。

图 2.8 所示为 LTE 下行链路架构中各种逻辑信道，传输信道和物理信道之间的关系。在单播模式，这里只有一种业务逻辑信道——专用业务信道（DTCH）——和 4 种控制逻辑信道：广播控制信道（BCCH），寻呼控制信道（PCCH）和专用控制信道（DCCH）。专用逻辑业务信道和所有逻辑控制信道，除了 PCCH 之外，共用下行链路公共信道。寻呼控制信道（PCCH）映射到寻呼信道（PCH），它和 DLSCH 一起构成了物理下行链路公共信道（PDSCH）。PDSCH 和 4 种其他的物理信道（PDCCH，物理下行链路控制信道；PHICH，物理混合自动重传请求指示信道；PCFICH，物理控制格式指示信道；PBCH，物理广播信道）从上层信道接收并提供所有单播模式所需的用户数据、控制信息和系统信息。

表2.6　LTE下行链路物理信道

下行链路物理信道	功能
物理下行链路公共信道（PDSCH）	单播用户数据业务和寻呼信道
物理下行链路控制信道（PDCCH）	下行链路控制信息（DCI）
物理混合式APQ指示信道（PHICH）	上行链路包的HARQ指示符和ACK/NACK
物理控制格式指示信道（PCFICH）	控制格式信息（CFI）包含解码PDCCH的必要信息
物理组播信道（PMCH）	组播广播单频率网络（MBSFN）操作
物理广播信道（PBCH）	当小区搜索时终端为了控制网络所需的系统信息

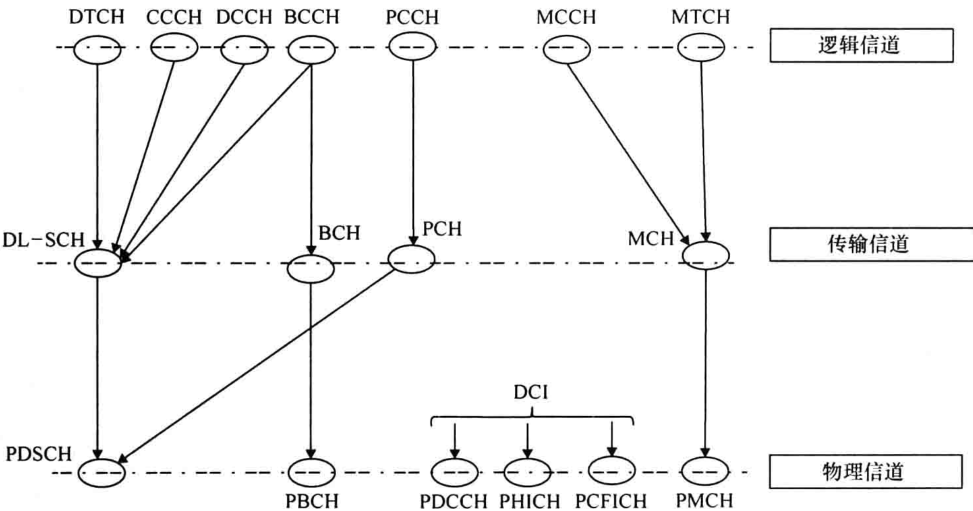


图2.8　LTE下行链路中逻辑信道、传输信道和物理信道间的映射

在多播/广播模式，存在多播业务信道（MTCH）和称为多播控制信道（MCCH）的控制逻辑信道。它们组成了传输信道，即多播信道（MCH）。最后，PMCH构成了MBMS模式的物理信道。

2.10.2　下行链路信道功能

PDSCH搭载下行链路用户数据，以传输块的方式从MAC层降至物理层。一般地，传输块以一个一个子帧发送，除了MIMO中空分复用，它可以在一个子帧里传输1～2个传输块。通过适应性调制和编码，调制符号映射到多个时－频资源网格，它们最终映射到多个发射天线进行发送。多径天线技术应用到每个子帧也需要根据信道条件适应性操作。

根据移动端反馈的信道质量情况，通过对每个子帧适应性调制、编码以及

MIMO，基站需要决策调制方案的类型，编码率和 MIMO 模式。终端的测量结果必须反馈到基站以帮助基站做调度决策保证传输质量。在每个子帧上，移动端需要关注基站的每一个发送资源块调度。这些必要的通信信息是每个子帧中使用的一组调度给用户的资源块、传输块长度、调制类型，编码率和 MIMO 类型信息。

为了维护基站和移动终端之间的通信，PDCCH 定义在每个 PDSCH 信道上。PDCCH 主要包括保证每个终端成功接收、平衡，解调和解码数据包的调度决策。因 PDCCH 信息在 PDSCH 开始解码之前必须读取并解码，PDCCH 在下行链路占据每个子帧开始的几个 OFDM 符号。PDCCH 在每个子帧上占据多少个 OFDM 符号（一般是 1 ~ 4 个），取决于多个因素，包括带宽、子帧索引，以及是否使用单播还是组播服务。

搭载在 PDCCH 上的控制信息即 DCI。根据 DCI 格式的不同，一组资源元素（比如几个 OFDM 符号需要搭载它）也会不同。LTE 标准定义了 10 种可能的 DCI 格式。表 2.7 总结了可用的 DCI 格式以及它们通常使用的实例。

每个 DCI 格式包括了如下几种控制信息：资源分配信息，如资源块长度和资源分配时间；传输信息，如多径天线配置信息、调制类型、编码率和传输块有效长度；和 HARQ 有关的信息，包括过程号、冗余版本和新数据的信号可用性指标。表 2.8 总结了 DCI 格式 1 的内容。

表 2.7 LTE 下行链路控制信息（DCI）格式和它的应用实例

DCI 格式	应用实例
0	上行链路调度分配
1	SISO 和 SIMO 模式下对一个 PDSCH 码字进行下行链路调度
1A	
1B	对 MIMO 模式 6 一个 PDSCH 码字非常简洁的下行链路调度
1C	
1D	多用户 MIMO 下，对一个 PDSCH 码字附带 MIMO 预编码和功率偏移信息进行简洁的下行链路调度
2	对闭环空分复用 MIMO 进行下行链路调度分配
2A	对开环空分复用 MIMO 进行下行链路调度分配
3	
3A	

表 2.8　DCI 格式 1 的内容

对象	PDCCH 的比特数	描述
资源分配数据头	1	指示选定资源分配类型 0 或 1
资源块分配	与资源分配类型有关	指示分配到终端的 PDSCH 资源
调制和编码方案（MCS）	5	指示使用调制和编码类型、传输快长度和分配资源数
HARQ 处理数	3（FDD）4（TDD）	指示 HARQ ID 用于非同步停止等待协议
新数据指示码	1	指示当前包是否时新包或重传包
冗余版本	2	指示 HARQ 增量冗余状态
PUCCH TPC 命令	2	指示与发射功率匹配的 PUCCH 功率控制命令
下行链路分配索引	2	（只对 TDD 模式）指示上行链路 ACK/NACK 捆绑的下行链路子帧数

PCFICH 用以定义在一个子帧里 DCI 使用的 OFDM 符号数量。PCFICH 信息被映射到每一个子帧开始的 OFDM 符号内特定的资源元素上。PCFICH 的可能值（1、2、3 或 4）取决于带宽、帧结构和子帧索引。当带宽大于 1.4MHz，PCFICH 码会占用 3 个 OFDM 符号。当带宽为 1.4MHz 时，因为资源块的数量很少，PCFICH 可能会需要 4 个符号用于搭载控制信号。

除了 PDCCH 和 PCFICH 控制信道之外，LTE 定义了另一个控制信道，即物理 HARQ 指示信道（PHICH）。PHICH 包括下行链路上接受到包的通知应答信息。上行链路包发送过程中，当预测到延时发生时，UE 会接收到一个由 PHICH 资源块携带的通知。PHICH 的时长由上层决定。在标准时长下，PHICH 只存在于子帧的第一个 OFDM 符号。在扩展时长下，它将占用最开始的 3 个子帧。

PBCH 携带主信息块（MIB），它包含了用于小区搜索时的基本物理层系统信息和小区专有信息。移动终端在正确的得到 MIB 之后，将会读取下行链路控制和数据信道，进行接入系统所需的必要的操作。PBCH 上的 MIB 以每 40ms 周期发送，对应 4 个无线帧，位于每个帧的第一个子帧。MIB 包括 4 部分信息。第 1，2 部分信息包括下行链路带宽和 PHICH 配置。下行链路带宽由 6 个下行链路资源块编号值中的一个表示（6、15、25、50、75 或 100）。如前面所讨论的，这些值对应的资源块编号直接一一映射到 1.4MHz、3MHz、5MHz、10MHz、15MHz 和 20MHz 这些带宽。PHICH 中 MIB 的配置部分指定了时长和 PHICH 的数量，如前文所述。在频域上，PBCH 永远对应每一个无线帧中第一个子帧内前 4 个 OFDM 符号，占据以 DC 子载波为中心的 72 个子载波。在下面关于物理信号的描述中，我们将会完整描述 LTE 标准中这些帧结构的内容。

2.10.3 上行链路物理信道

表 2.9 总结了 LTE 上行链路物理信道。物理上行链路公共信道（PUSCH）搭载用户数据从用户端发送。物理随机接入信道（PRACH）用于 UE 发送出随机接入报头，初始化接入网络。物理下行链路控制信道（PUCCH）携带 UCI，包括调度请求（SR）、发送成功或失败的通知（ACK/NACK）、内有信道质量指示的下行链路信道测量报告、预编码矩阵信息（PMI），以及秩指示（RI）。

表 2.9 LTE 上行链路物理信道

上行链路物理信道	功能
物理上行链路公共信道	上行链路用户数据业务
物理上行链路控制信道	上行链路控制信息
物理上行链路接入信道	通过随机接入报头初始化接入网络

图 2.9 所示为 LTE 上行链路结构中逻辑信道、传输信道和物理信道之间的关系。在逻辑信道层内有专用业务信道（DTCH）和两个逻辑控制信道，一个公共控制信道（CCCH），和一个专用控制信道（DCCH）。这三个信道构成了上行链路公共信道（UL－SCH），它是传输信道层。最后，物理上行链路公共信道（PUSCH）和物理上行链路控制信道（PUCCH）构成了物理信道层。传输信道中的随机接入信道（RACH）也映射到物理随机接入信道（PRACH）。

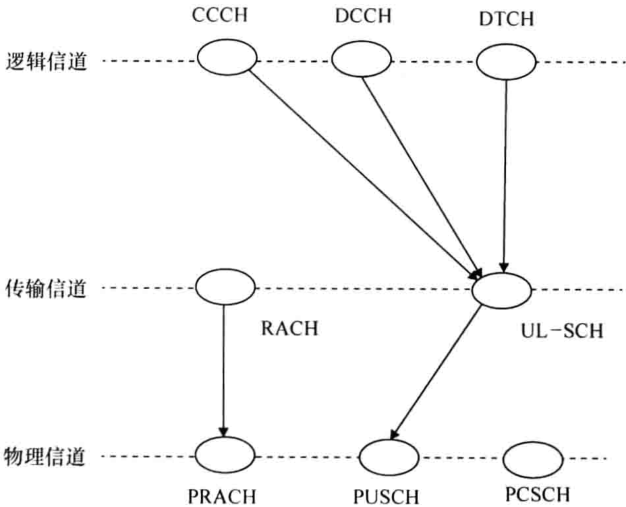


图 2.9 LTE 上行链路逻辑、传输和物理信道的映射关系

2.10.4 上行链路信道功能

PUCCH 搭载三种控制信息：用于下行链路传输的 ACK/NACK 信号、调度请求（SR）指示符，和下行链路信息的反馈，包括 CQI、PMI 和 RI。

下行链路信息的反馈与下行链路的 MIMO 模式有关。为了保证下行链路中 MIMO 正确正常工作，每一个终端必须测量无线链路质量并向基站报告信道特征参数。这一工作实质上反映了 PUCCH 中 UCI 的信道质量测试功能。

CQI 是下行链路无线信道质量测量的指示符，它由 UE 生成并传输给基站用于随后的调度。CQI 允许 UE 向基站提交一系列最佳的调制方案和编码率匹配当前无线链路质量。CQI 信息包括 16 个调制方案和编码率组合。更高的 CQI 值对应更高阶调制和更高的编码率。宽带 CQI 用于所有资源块形成带宽，子带 CQI 分配特定的 CQI 值对应特定的资源块。更上层配置决定了速率、周期，或终端 CQI 测量的频率。

PMI 是预编码矩阵的指示符，用于给定无线链接中的基站一侧。PMI 值反映了 1 个，4 个或 8 个发射天线配置对应的预编码表。RI 表示可用的发射天线数量，它由信道质量估计生成，影响相邻接收天线相关性测量。在后面章节，我们将讲解 LTE 标准中 MIMO 的模式。在这些章节中，CQI、PMI 和 RI 指示符的作用将完全清晰。

2.11 物理信号

物理信号多种多样，包括参考和同步信号，在公共物理信道内传输。物理信号映射对应 PHY 特定的资源元素但并不携带上层信息。下面我们详细说明 LTE 各信号。

2.11.1 参考信号

在频域上的信道相关性调度是 LTE 最有特点的部分。举例说明，为了表现反映真实信道质量的下行链路调度，移动终端必须为基站提供信道状态信息（CSI）。CSI 由测量下行链路中参考信号生成。参考信号由发射端和接收端的同步序列生成器生成。这些信号在时 - 频网格中被放置于特定的资源元素。LTE 定义了一系列下行链路和上行链路参考信号类型。我们将在下面讨论。

2.11.1.1 下行链路参考信号

下行链路参考信号提供了信道测量功能，它用于平衡和解调控制信息和数据信息。它们也辅助 CSI 测量（如 RI、CQI 和 PMI），用于信道质量反馈。LTE 定义 5 种参考信号类型：小区特定参考信号（CSR），解调参考信号（DM - RS 以及其他如 UE 特定参考信号等），信道状态信息参考信号（CSI - RS），MBSFN 参

考信号和位置参考信号。

CSR 为所有小区用户和所有下行链路子帧共用。DM－RS 用于下行链路多用户传输模式 7、8、9。如其名字所示，它用于小区内每一个单独移动终端信道估计。CSI－RS 在 LTE 第 10 发布版中第一次出现。它的主要功能是减轻多于 8 个天线情况下 CSR 和 CSI 测量的密度问题。因此，CSI－RS 只用于多用户下行链路传输模式 9。MBSFN 参考信号用于组播/广播服务中的相干解调。位置调制信号，第一次出现在 LTE 第 9 发布版中，用于协助多小区测量以估计给定终端的位置。在这一节，我们将会更详细讲解前 3 个参考信号。

1. 小区特定参考信号

CRS 存在于每一个下行链路子帧和频域资源块，因此它涵盖了全部小区带宽。除使用非码书预编码的传输模式 7、8、9 中 PMCH 和 PDSCH 之外，CSI 可在终端测量任意下行链路物理信道相干解调的信道估计。

CRS 也可在终端用以得到 CSI。在终端 CRS 上进行的测量，如 CQI、RI 和 MI，也是小区选择和切换判决的基础。

2. UE 特定参考信号

DM－RS，或 UE 特定参考信号，只在下行链路传输模式 7、8 或 9 使用。这三个模式中，CSR 不用于信道估计。LTE 第 8 发布版第一次引入 DM－RS，支持一个信号层。在 LTE 第 9 发布版中，它支持两个信号层。在随后的扩展版本 LTE 第 10 发布版中，它同时支持 8 个参考信号。

当使用一个 DM－RS，有 12 个参考符存在于一对资源块中。随后我们会讲到，当一个资源元素在任意一个给定天线发射参考信号时，CSR 需要所有其他天线端口的频谱零点或不使用的资源元素。这是 CSR 和 DM－RS 的一个主要区别。当两个天线使用两个 DM－RS 时，12 个参考符在两个天线端口传送。参考信号间的干扰可通过为每一组相邻的参考符号生成一个相互正交的图形来减轻。

3. CSI 参考信号

CSI－RS 用于 4 个或 8 个天线的情况。CSI－RS 在 LTE 第 10 发布版中引入。它是 LTE 传输模式 9 中 DM－RS 的辅助功能。当 DM－RS 用于提供信道估计时，CSI－RS 用于得到 CSI。为了减小一个资源网格存在两种类型参考信号的花费，CSI－RS 的瞬时分辨率被减小了。这使得系统不能捕捉信道条件的快速变化。因 CSI－RS 只用于 8 个天线 MIMO 配置，以及该配置只用于低移动性条件，CSI－RS 的低瞬时分辨率不会造成问题。

2.11.1.2　上行链路参考信号

LTE 标准定义两种上行链路参考信号：DM－RS 和探测参考信号（SRS）。这两种参考信号基于 Zadoff－Chu 序列。Zadoff－Chu 序列用于生成下行链路主同步信号（PSS）和下行链路前导信号。不同 UE 的参考信号由基本序列的不同循

环移位参数区分。

1. 解调参考信号

DM－RS 作为上行链路资源网格的一部分由 UE 发送。它用于基站接收端平衡和解调上行链路控制（PUCCH）和数据（PUSCH）信息。对于 PUSCH，当使用普通循环前缀时，DSR 信号在每个 0.5ms 时隙中前 4 个 OFDM 符号占用所有资源块。对于 PUCCH，DSR 的位置取决于控制信道的格式。

2. 探测参考信号

在上行链路中，SRS 用于基站估计不同频率的上行链路信道响应。信道状态估计用于上行链路信道相关性调度。调度器可以在信道响应较好的上行链路带宽上集中分配用户数据。SRS 还有另外一个应用，即当下行链路和上行链路信道交互或公用时，进行定时估计和控制下行链路信道条件，如 TDD 模式。

2.11.2 同步信号

除了参考信号之外，LTE 还定义了同步信号。下行链路同步信号用于多个处理过程中，包括帧边界检测、确定天线数量、初始化小区搜索、相邻小区搜索和交接。LTE 定义了两种同步信号：主同步信号（PSS）和辅助同步信号（SSS）。

PSS 和 SSS 占用 DC 子波段周围的 72 个子波段。不过，FDD 模式下这些位置不同于 TDD 模式。在 FDD 帧中，它们使用子帧 0 和 5，彼此相邻。在 TDD 帧中，它们并不相邻。SSS 信号位于子帧 0 和 5 的最后一个符号，PSS 位于一个特定帧的第一个 OFDM 符号。

同步信号与 PHY 小区识别有关。LTE 定义了 504 个小区识别码，分为 168 个组，每个组包括 3 个特殊识别码。PSS 搭载特殊识别码 0、1 或 2，而 SSS 搭载组识别码 0～167。

2.12 下行链路帧结构

LTE 定义了两种下行链路帧结构。第一种帧用于 FDD，第二种用于 TDD。每种帧由 10 个子帧组成，每个子帧由时－频资源网格描述。我们知道一个资源网格包括三个组成部分：用户数据、控制信道和参考、同步信号。现在我们可以解释这些组成部分的具体位置和它们如何组成了 LTE 子帧中的资源网格。本书将聚焦 FDD 帧结构和第一种帧。

图 2.10 所示为第一种帧的结构。帧时长为 10ms，包括 10 个 1ms 的子帧，它们从 0～9 编号。每个子帧可以氛围 2 个 0.5ms 时长的时隙。每个时隙包括 7 个或 6 个 OFDM 符号，取决于循环前缀类型。DCI 位于每个子帧的第一个时隙。DCI 搭载了 PDCCH，PCFICH 和 PHICH。他们合起来占据了每个子帧的前三个

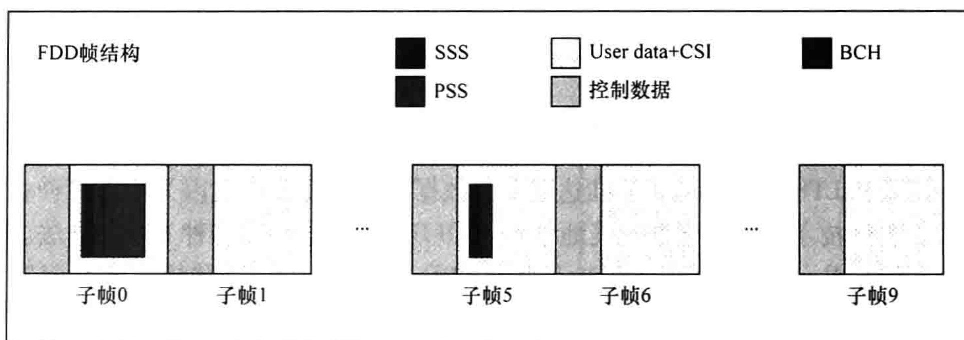


图 2.10 下行链路 FDD 子帧结构

OFDM 符号。这一区域也就是 L1/L2 控制区，包含了从层 2（MAC 层）向层 1（PHY 层）传送的信息。

PBCH 包含的 MIB 位于子帧 0，而 PSS 和 SSS 位于子帧 0 和 5. PBCH 信道与 PSS 和 SSS 信号占用以 DC 子载波为中心的 6 个资源块。此外，CSR 作为一种特殊的时域和频域图形占据每个子帧所有的资源快。CSR 信号的这个图形的位置取决于 MIMO 模式和可用天线数。我们在下文将很快讨论这些。子帧内的其他的资源元素分配给用户业务数据。

2.13 上行链路帧结构

上行链路帧结构在一定程度上与下星莲路相似。它也包括 1ms 子帧，分为两个 0.5ms 时隙。每个时隙包括 6 个或 7 个 SC－FDM 符号，取决于循环前缀类型。带内资源块用于接收数据资源元素（PUSCH）以减少带外发射。不同用户使用不同的资源块，以保证同一小区多用户间的正交性。数据传输可以跳跃时隙边界从而提供了频率分隔。控制资源（PUCCH）位于载波段的边缘，由中间时隙跳跃提供频率分隔。信号解调所需的参考信号散布于数据和控制信道。图 2.11 描述了上

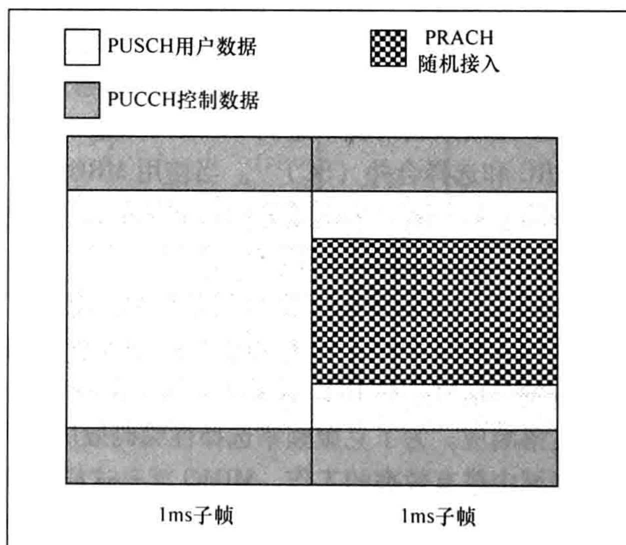


图 2.11 上行链路帧结构

行链路帧结构。

2.14 MIMO

LTE 和 LTE - Advanced 标准达到最大数据速率一定程度上因为使用多种多天线或 MIMO 技术。LTE 标准完美地结合了 OFDM 传输结构和多种 MIMO 方法。正如此，LTE 是一个 MIMO - OFDM 系统。如前文所言，OFDM 传输方案在每路天线上构建资源网格，生成 OFDM 符号，并发送它们。对一个 MIMO - OFDM 系统，这一过程由多个发射天线重复完成。随着 OFDM 符号搭载多个资源网格在多个发射天线上发送，它们被合并在一起发往每个接收天线上。MIMO 接收器的任务也因此是分离合并在一起的信号，根据资源元素的接收估计，解析每个发射天线发送的源资源元素。

多天线技术依托接收器或发射器使用多个天线传输以及其先进的信号处理技术。虽然多天线技术增加了执行的可计算性复杂度，但它可以达到提升系统性能，包括提升系统容量（换句话说就是一个小区网络容纳更多用户）和提升覆盖率或更大范围小区传输的可能性作用。发送端或接收端多天线的实用性体现在多个不同方面，实现多个不同目标。

2.14.1 接收分集

多天线最简单和最常用的配置是在接收端使用多天线（见图 2.12）。应用于接收分集的最重要的算法就是最大比合并。它应用于 LTE 传输的模式 1，该模式基于单天线传输。该模式也称为 SISO（单输入单输出），它只是用一根接收端天线或在多接收天线情况下进行 SIMO（单输入多输出）。接收端使用另种合并方法：MRC 和选择合并（SC）[2]。当使用 MRC 时，我们合并多路接收信号（一般通过平均它们）找到发射信号的最似然估计。当使用 SC 时，只有最高 SNR 的接收信号被采用用以估计发射信号。

MRC 是一种非常优秀的 MIMO 技术，当在衰落信道中，交调信号数量多并保持同样大的强度时，MRC 可以 z 在平坦衰落信道中工作得很好。实际上，绝大部分宽带信号，如 LTE 中所定义的，回收时间扩展效应的影响，导致频率选择性衰落响应。为了克服频率选择性编码效应，我们必须使用线性平衡技术并使其在频域中耕有效率的工作。MIMO 就是这样一种很好抵消劣化的技术。我们将会在下面讨论它。

2.14.2 发射分集

发射分集即在发射端使用多天线通过发射相同信号的随机版本。这一类

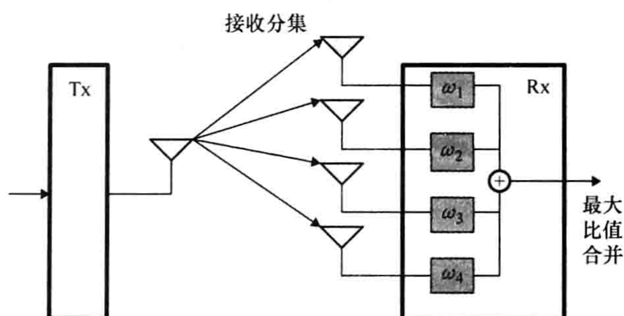


图 2.12 MIMO 接收分集

MIMO技术一般为空－时区块编码（STBC）。使用 STBC 调制时，符号映射到时域和空域（发射天线）捕捉多发射天线的分集。

空－频区块编码（SFBC）时一种和 STBC 非常相关的技术，它作为发射分集技术引入 LTE 标准。这两种技术的主要区别在于 SFBC 在天线（空域）和频域编码而不是在天线（空域）和时域编码，而 STBC 正相反。SFBC 的流程简图如图 2.13 所示。

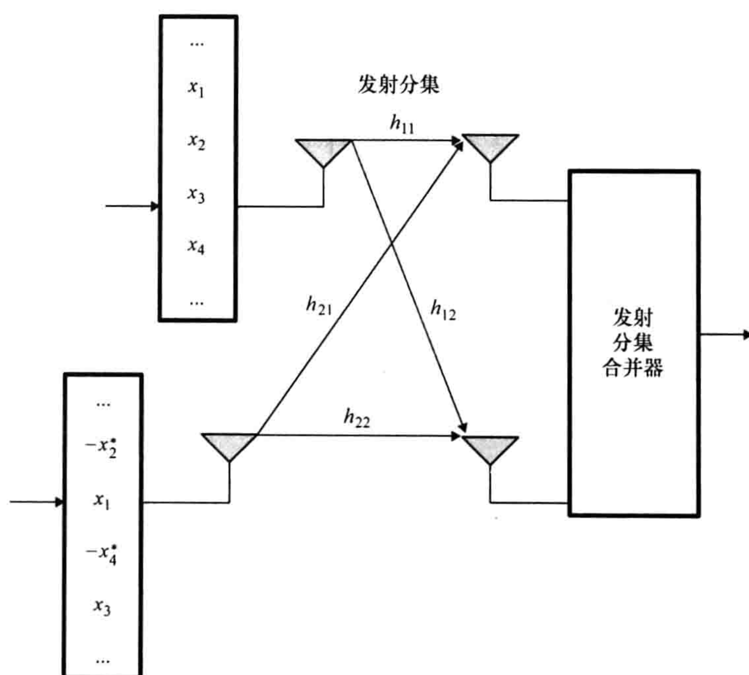


图 2.13 MIMO 空－频区块编码

LTE 中，第二种传输模式基于发射分集。SFBC 和频率转换发射分集（FSTD）分别用于两个和四个天线发射。发射分集并不会对数据速率有提升作用，它只是增加了对信道衰落影响的可靠性并增加了链路质量。其他 MIMO 模式—特别是空分复用—则直接增加了数据速率。

2.14.3　空分复用

在空分复用情况下，完全独立的数据流在每个发射天线上同时被发射。应用空分复用可使系统与发射天线端口数量等比例的提高数据速率。同时，在同一频率载波上，不同调制符号通过不同天线发射。这意味着空分复用可以直接提升带宽效率，提高系统的带宽利用率。空分复用的这一好处只在多发射天线彼此不相关时才能体现。空分复用可以在通信链路自然存在多路衰落情况下提高性能。因多路衰落可以在每个接收天线端口与接收信号去相关，在多路衰落信道使用空分复用可以事实上提高性能。

空分复用的所有优势只在系统与发射和接收天线有关的线性平衡可解时才能体现。图 2.14 所示为 2×2 天线配置下的空分复用。在每个子载波上，调制符号 s_1 和 s_2 通过两个发射天线发射。在同一子载波频率 r_1 和 r_2 上的接收符号可以考虑为 s_1 和 s_2 的线性合并加权信道矩阵 H，并附加 AWGN（加性高斯白噪声）项 n_1 和 n_2 的结果。MIMO 方程可表示为

$$\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \tag{2.5}$$

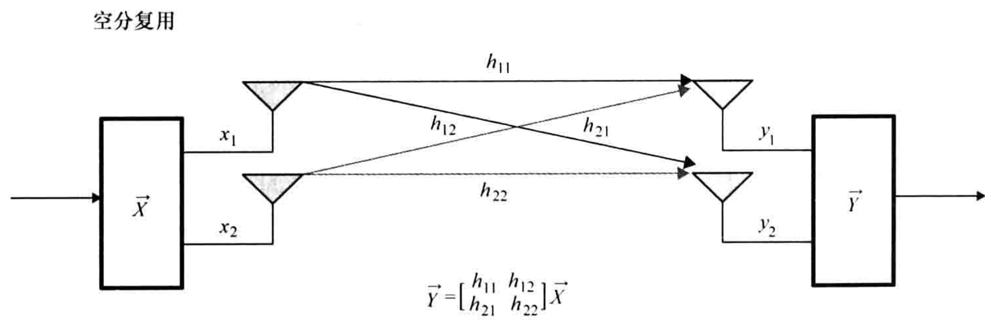


图 2.14　MIMO 空分复用

MIMO 信道矩阵 $\boldsymbol{H}$ 包括每个子载波信道频率响应 H_{ij}，i、j 意为任意传输信道 i 和接收天线 j 的组合。当矩阵符号由任意个发射和接收天线生成时，上式可表示为

$$\vec{r} = \boldsymbol{H}\vec{s} + \vec{n} \tag{2.6}$$

式中 $\vec{s}$为发射端发射信号的 M 阶向量：$\vec{s} = [s_1, s_2, \cdots, s_M]$；$\vec{r}$和$\vec{n}$为 N 阶向量，对应接收信号和相应的噪声信号：$\vec{r} = [r_1, r_2, \cdots, r_M]$；$\vec{n} = [n_1, n_2, \cdots, n_M]$。

当向量$\vec{s}$的元素属于单一用户时，此单一用户的数据流由数个天线复用。此即单用户多输入多输出（SU - MIMO）系统。当不同天线复用不同用户数据流时，系统为多用户多输入多输出（MU - MIMO）系统。SU - MIMO 系统根本上提升了给定用户的数据速率，而 MU - MIMO 系统提升了包括多个终端小区的整体容量。

有关空域服用系统操作的一个最基本的问题是是否相关性 MIMO 方程可解并有唯一解。这个问题和相关性 MIMO 信道矩阵的特异性以及它是否可以取反有关。当接收信号或多个接收天线相关，信道矩阵 $\boldsymbol{H}$ 会含有现行相关的行或列。在这种情况下，新到矩阵会因为秩小于维数而无法取反。因此，秩估计对于空分复用是必要的，它决定了任意给定信道条件下空分复用操作是否可行。矩阵秩的绝对值表示可以成功复用的发射天线数量。LTE 技术中，矩阵的秩也表示了空分复用 MIMO 模式下的层数。

在闭环 MIMO 操作中，信道矩阵的秩由移动终端计算并通过上行链路控制信道送往基站。假如这个信道被认为没有满秩，则只有减小独立数据流的数量在下行链路中完成空分复用。这一特点，即秩适应性，是适应性 MIMO 方案的一部分以及 LTE 标准中其他适应性特征的补充。

2.14.4 波束赋形

在波束赋形过程中，发射天线可以形成所有天线发射图形（或波束）来达到在移动终端方向全天线增益最大化。波束赋形构成了下行链路 MIMO 传输模式 7 的基础。

应用波束赋形技术可以实现信号功率随发射天线数量成比例增长。一般来说，波束赋形依赖最少 8 个天线构成的天线阵列工作[3]。波束赋形由天线阵列中不同元素应用不同的复变增益（或称权重）执行。所有的传输波束可以指向不同方向，这一过程由在不同天线信号上进行不同的相移完成，如图 2. 15 所示。

LTE 标准即没有定义天线阵列内天线数量，也没有定义在每个天线阵列元素间校准复变增益的算法。LTE 定义天线端口 5，以此表示使用波束赋形的虚拟天线端口。UE 专有参考信号用来在波束赋形 MIMO 模式 7 下进行信道估计。更高层要求对移动终端使用 UE 专有参考信号。因正交参考信号在同一组资源网格相

互生成调度，不同 UE（移动终端）可以解析它们分配到的参考信号，并在平衡和解调中应用它们。

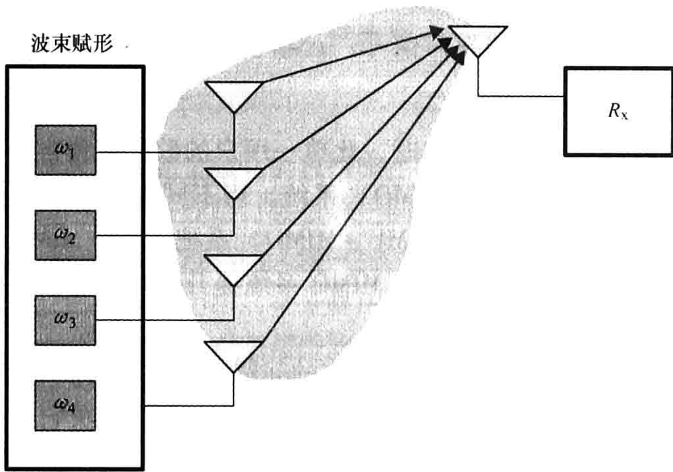


图 2.15　MIMO 波束赋形

2.14.5　循环延迟分集

循环延迟分集（CDD）时 LTE 标准中结合开环空分复用的另一种分集形式。CDD 对任意给定时刻不同天线上发射信号的向量或块进行循环移位。其效果如同使用一个已知的预编码器。如此，CDD 与块传输方案如 OFDM 和 SC - FDM 非常匹配。在 OFDM 传输情况下，比如，时域的循环移位对应频域上频率相关性相移。因相移在频域上——也就是预编码矩阵——可知和可预知，CDD 应用于开环空分复用以及在高移动率情况下优化预编码矩阵的闭环反馈无法完成的情况。应用 CDD 主要作用就是在接收端经验性的引入一个虚拟的频率分隔。我们可以很轻易的在多于 2 个发射天线上扩展 CDD，在每个天线上使用不同的循环移位。

2.15　MIMO 模式

表 2.10 总结了 LTE 传输模式以及与其有关的多天线传输方案。模式 1 使用接收分集，模式 2 基于发射分集。模式 3 和 4 为单用户空分复用，分别基于开环或和闭环预编码。模式 3 也使用 CDD（如前文所述）。

LTE 模式 5 定义了一个非常简单的多用户 MIMO，基于模式 4 并将最大层数设定为 1。模式 6 为模式 4 的特殊情况，它使用波束赋形，并将最大层数设定为

2。LTE 模式 7～9 为不使用码书的空分复用，层数分别为 1、2，4～8。LTE - Advanced（第 10 发布版）引入模式 8 和 9 大大提升了下行链路 MU - MIMO 性能。如模式 9 支持 8 个天线在 8 个层传输。这些进步也直接来自于引入新的参考信号（CSI - RS 和 DM - RS），允许无码书的预编码并接收低开销双码书结构[4]。

表 2.10 LTE 传输模式与其所对应的多天线传输方案

LTE 传输模式	
模式 1	单天线传输
模式 2	发射分集
模式 3	开环码书预编码
模式 4	闭环码书预编码
模式 5	传输模式 4 的多用户 MIMO
模式 6	闭环码书预编码的单用户特殊情况
模式 7	发布版 8 支持单用户基于波束赋形的非码书预编码
模式 8	发布版 9 支持两用户的非码书预编码
模式 9	发布版 10 支持最多 8 用户的非码书预编码

2.16 物理层数据处理

为了理解 LTE 的物理层，我们归纳了以下一系列 PHY 层操作。首先，描述信道编码、绕码，调制以生成调制符号。然后描述调制信号映射到资源网格的每一步，包括映射用户数据、参考信号和控制数据。随后，定义允许多天线传输的 MIMO 模式。不同的 MIMO 算法对应指定层的映射，即每个帧使用多少发射天线，以及调制字节映射到所有发射天线资源网格之前，如何使用预编码传输。

2.17 下行链路数据处理

发射端一系列信号处理操作可以分为传输块处理与物理信道处理。3GPP 完整定义了复用和信道编码[5]，以及物理信道和调制[3]的处理栈。基带信号处理链可归纳为 DLSCH 和 PDSCH 的组合，如下所示：

—添加传输块 CRC（循环冗余检查）；

—码块分割和码块 CRC 添加；

—1/3 码率的 Turbo 编码；

—以要求的编码率进行速率匹配；

—链接码块生成码字；

—对每个码字的位进行绕码并在物理信道发送；

—调制绕码位生成复调制符号；

—映射复调制符号到一个或多个传输层；

—为了在天线端口传输，在每个传输层上对复调制符号进行预编码；

—映射复调制符号到每个天线端口的资源元素；

—在每个天线端口上生成复时域 OFDM 信号。

图 2.16 所示为 OFDM 信号发送之前，处理 MAC 层向 PHY 层发送的传输块信号的过程。

每个 LTE 下行链路组成部分将会在第 4～7 章详细描述。在第 4 章，我们详细说明 DLSCH 处理，绕码和调制映射机能。在第 5 章，我们详细说明下行链路 OFDM 多载波传输方案。在第 6 章，我们回顾标准中若干个 MIMO 执行过程。在第 7 章，我们描述应用于多个控制信道中随信道条件进行资源动态调度的链路自适应原理。

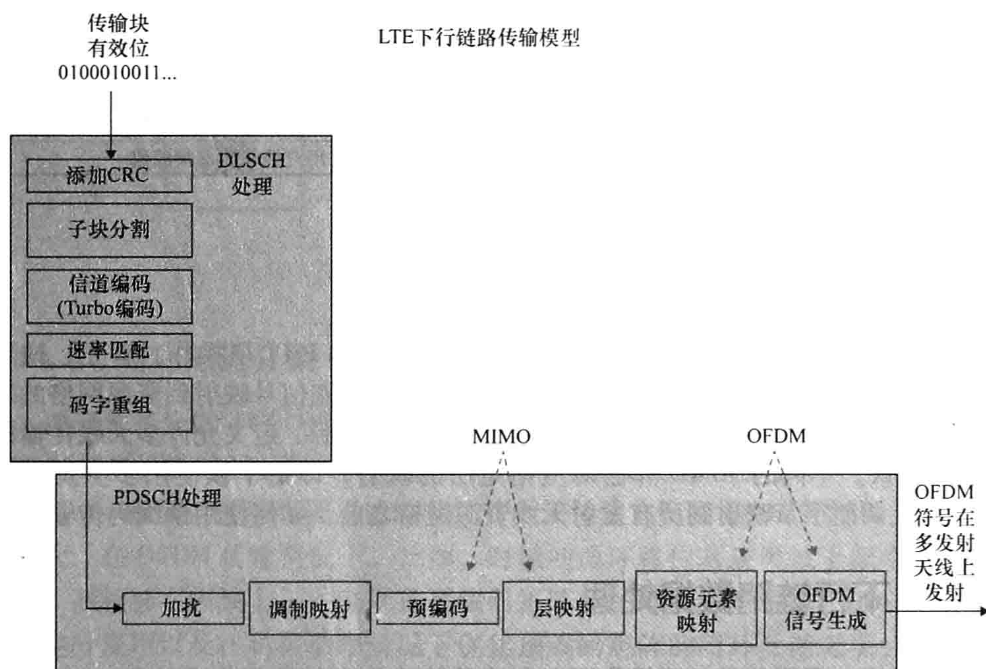


图 2.16　下行链路 DLSCH 和 PDSCH 的信号处理链

2.18　上行链路数据处理

上行链路信号处理过程可分为 ULSCH 和 PUSCH，如下所示：

—添加传输块 CRC；

—码块分割和码块 CRC 添加；

—1/3 码率的 Turbo 编码；

—以要求的编码率进行速率匹配；

—链接码块生成码字；
—绕码；
—调制绕码字位成复调制符号；
—映射复调制符号到一个或多个传输层；
—对复符号进行 DCT 传输预编码；
—对复调制符号进行预编码；
—映射复调制符号到资源元素；
—在每个天线端口上生成复时域 SC－FDM 信号。

图 2.17 所示为 SC－FDM 信号发送之前，处理 PHY 层接收信号的过程。3GPP 完整定义了复用和信道编码[5]，以及物理信道和调制[3]的处理栈。

在这一节中，我们将会描述上行链路传输的两个不同的组件：基于 DFT－预编码 OFDM 的 SC－FDM 和 MU－MIMO。

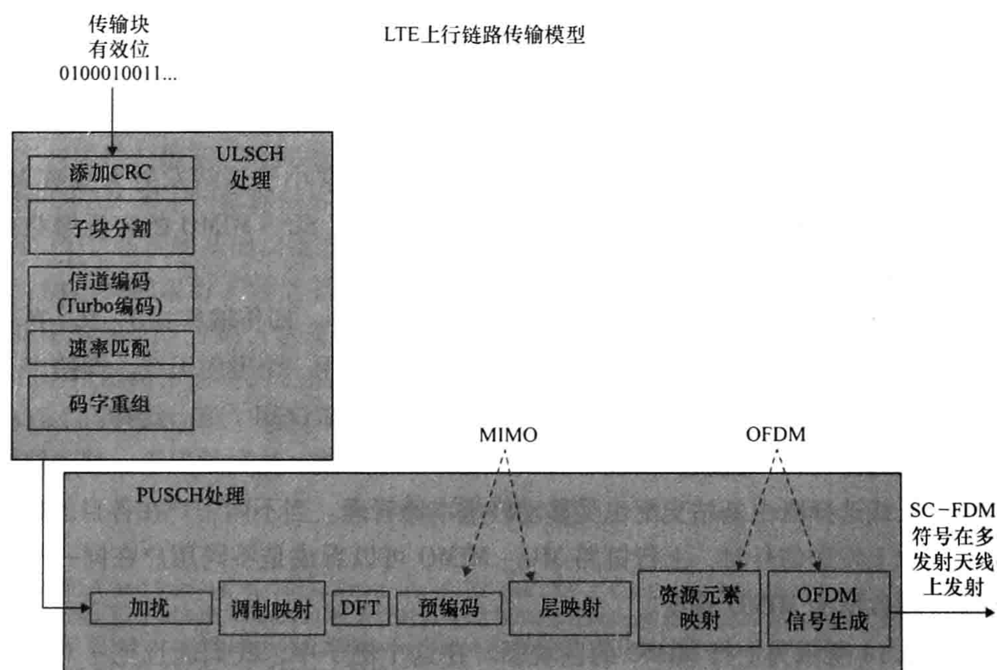


图 2.17 上行链路 ULSCH 和 PUSCH 信号处理链

2.18.1 SC－FDM

在 LTE 中，在调制符号上使用的是一种基于 DFT 应用的特殊预编码，用于在频域生成 SC－FDM 信号。注意 SC－FDM 信号生成过程与 OFDM 相同，除了引入附加的 M 点 DFT 之外。通常，DFT 计算的计算效率小于 FFT。不过我们可

以找到高效执行素数长度 DFT 运算的方法。这就是为什么 LTE 定义 M 点 DFT 的原因，其长度可以为 1、3 或 5（它们都是素数）。

在下行链路传输中，随着编码、绕码和映射资源元素前的调制，基于 DFT 的预编码用于所有层的调制符号。DFT 变换符号随后映射到 IFFT 操作和添加循环前之前的缀频率子载波，并最终生成 SC - FDM 信号。任意个体哟过户发送的数据符号作为 SC - FDM 符号必须相邻且平均的分布在资源网格上。

以上局部 DF 预编码符号在资源网格的映射意味着所有分配在频率上相邻。当导频信号相邻且信道估计可以使用简单差值时，信道估计性能可以达到一个可接受的程度。另外，基于相邻资源块图案的多用户频谱复用非常简单。分散式映射，另一方面，意味着在频率上分散给定带宽。这种类型的映射可以很好地测量频率分隔。不过，因为导频信号也被其分散，将牺牲部分信道估计性能。在频谱上复用所有用户也会导致复杂的分散式映射。如上所述，分散式或局限式频率分配表现了频率分隔与性能的典型折中。

2.18.2 MU - MIMO

在移动系统中，移动终端的接收天线数量 N 常常小于基站发射天线数 M。因 MIMO 系统的容量增益由参数 min（M，N）约束，SU - MIMO 的容量增益被接收端接收天线数量 N 所限制[4]。

在下行链路传输中，这个问题由 MU - MIMO 处理，即传输模式 7 ~ 9。在上行链路，LTE 第 8 发布版本只支持在移动端以此只是用一个发射天线，即使多天线存在。减小费用、功耗和移动终端硬件复杂度的需求决定了这一选择。

天线选择可以一次从多个发射天线中选择一个天线。这种情况下，移动终端的发射天线选择既由基站支配也受移动终端本地管理。当不同用户在各自的移动单元天线上发送信号时，上行链路 MU - MIMO 可以看成是不同用户在同一个资源块传输数据的 MIMO 系统。

图 2.18 所示为上行 MIMO 的区块图。在这个例子中，我们通过把一对移动单元的传输组对，设定了一些 MU - MIMO 组。基站调度每个 UE 的上行链路传输，在同一个子帧和同一个资源块上进行 MU - MIMO 编组。由于系统带宽上可用的资源块数量不同，编组可随注入功耗控制、个体信道质量和干扰情况实时调整。在我们的例子里，我们设定了两组用户，而 LTE - advanced 中总括 DM - RS 和 CSI - RS 参考信号可支持 8 个移动终端 MU - MIMO 公用同一个资源块。更多关于 MU - MIMO 的信息，请参考参考文献[4]。

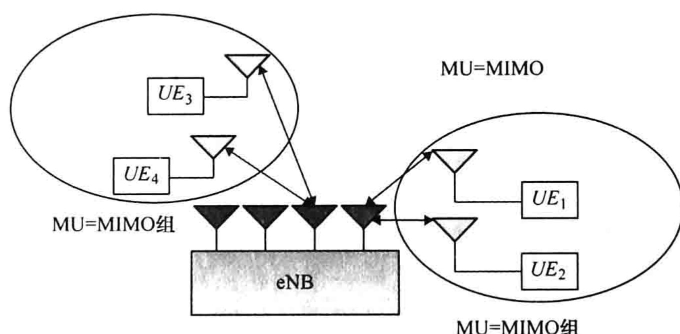


图 2.18 上行链路 MU－MIMO

2.19 本章小结

在本章中，我们学习了 LTE 标准中的 PHY 层协议。我们集中在一些 PHY 层模型中关键元素，以更深理解 PHY 层。首先，我们考察了 LTE 的空中借口，详述它的频带、带宽、时间帧和时－频结构。随后详细讲述了多载波方案：下行链路的 OFDM 和上行链路的 SC－FDM 传输。我们考察了 OFDM 资源块构成，它使理解 PHY 建模的基础。我们也讨论了上行链路和下行链路的帧结构。

随后我们遍历了用于上行链路和下行链路的物理信道和物理信号。介绍了 LTE 中的 MIMO 方案，它完整定义了若干个传输模式。最后，我们总结了下行链路和上行链路传输中的一系列操作。我们将在第 4 章到第 7 章中更详细讲述用 MATLAB 建模这些处理链。

参 考 文 献

[1] Ghosh, A. and Ratasuk, R. (2011) *Essentials of LTE and LTE-A*, Cambridge University Press, Cambridge.
[2] Dahlman, E., Parkvall, S. and Sköld, J. (2011) *4G LTE/LTE-Advanced for Mobile Broadband*, Elsevier.
[3] 3GPP (2011) Evolved Universal Terrestrial Radio Access (E-UTRA), , Physical Channels and Modulation Version 10.0.0. TS 36.211, January 2011.
[4] C. Lim, T. Yoo, B. Clerckx, B. Lee, B. Shim, Recent trend of multiuser MIMO in LTE-advanced, *IEEE Magazine*, 51, 3, 127–136, 2013.
[5] 3GPP (2011) Evolved Universal Terrestrial Radio Access (E-UTRA), Multiplexing and Channel Coding. TS 36.212.

3 MATLAB 通信系统设计

在本章中，我们介绍一些 MATLAB 的分析、设计、建模、仿真和执行以及验证通信系统的功能。我们将解决这样一个问题：MATLAB，这个高级编程语言以及它的设计仿真环境、软件工具箱扩展库是如何帮助研究者和实践工程人员开发移动和无线系统的?

3.1 系统开发流程

为了回答上面的问题，我们需要回顾若干个开发步骤：从早期研究和集成独立算法的算法设计，到系统原型，再到用仿真验证系统，检查系统的可实现性，评估资源消耗、内存、复杂度等，最后敲进软件或硬件代码执行这个设计。在执行步骤之前——也就是系统级资源评估——需要一些软件代码以进行系统级仿真。它也包含一些如数据类型和内存实际约束，也如复杂度的权衡。系统级代码可以作为硬件执行的基础，其目的是足够整合，使执行器进程可以创建一个软件仿真的比特级精度模型。它既能为可在数字处理器（Digital Signal Processor，DSP）上执行的汇编代码，又能为可在现场可编程门阵列（Field-Programmable Gate Array，FPGA）或专用集成电路（Application-Specific Intergrated Circuit，ASIC）上执行的硬件描述语言（Hardware Description Language，HDL）代码。在这一过程中，我们必须不断地监视新添加进模块中的细节，以确保设计仍然符合研发要求。

3.2 挑战和能力

当我们从制定规范开始到实现设计，我们面对很多挑战。这些挑战包括：

1）为实现设计制定蓝图，包括从协议的书面描述转换成软件模型；

2）为标准灵活定义的接收端引入先进独特的算法；

3）执行软件模型以动态评估系统级性能；

4）为处理大数据量提高仿真速度；

5）衔接流程中的每一步。

MATLAB 和它的工具箱可以帮助我们解决下面这些挑战。

1）数字信号处理和高级线性代数，作为 LTE 标准的基础，构成了 MATLAB 的核心优势。LTE 标准架构可以逐步由 MATLAB 程序综合而成。

2）通信系统工具箱提供了强大的 MATLAB 工具建立通信系统模型。超过 100 个调制、信道建模、误码检出、MIMO（多输入多输出）技术和平衡等算法，使我们可以设计通信系统而不是设计软件。工具箱也包括很多协议实例以使我们快速上手。

1）MATLAB 和 Simulink 是动态和大规模仿真的理想环境；

2）MATLAB 可以为仿真提速；

3）MATLAB 可以衔接设计流程中的每一步，通过：

① C/C + + 和 HDL 代码自动生成；

② 半实物验证。

我们可以把这些能力分为四类：算法开发、建模和仿真、仿真加速和衔接执行。在本章中，通过下面 MATLAB 和其核心的 Simulink 的快速介绍，我们会介绍三类能力：

1）建模和仿真工具；

2）仿真加速工具；

3）与执行衔接的工具。

建模和仿真能力，包括一系列工具箱，可以使用户建立通信标准的仿真模型，包括无线和移动标准。通过运行这些仿真模型，可使设计者评估系统整体和单一算法的性能，测定信道劣化和其他实时环境的影响。

3.3 关注点

本书的关注点是 LTE 的 PHY 层（物理层）MATLAB 建模。例如，我们会讨论 FDD 模式下的 LTE 建模和仿真。通过一点点程序的改进，读者就可以轻松读懂描述 TDD 模式的 MATLAB 程序。我们不会涉及所有的控制平台处理、漫游或随机接入、多媒体广播帧的内容，也不会涉及与组播模式或多用户 MIMO 有关所有细节的 MATLAB 程序。我们不会关注如分配小区里移动终端这样的一般问题，我们会全部关注用户平台数据处理的细节。

3.4 目标

从 LTE 最简单的组件开始（如调制），我们会创建一系列的 MATLAB 程序，循序渐进地增添如绕码、信道编码这些组件构建信号处理链。在每一步，我们会估计计算性能指标如比特误码率（BER）以确保组件组合被正确建模。我们会继续这一过程，创建 OFDM 和 MIMO 的 MATLAB 程序。我们也会建立多个子程序帮助匹配模型和 LTE 标准。在这一过程的最后，我们会得到一个 MATLAB 程

序和 Simulink 模型表现 LTE 下行链路模式的重要信号处理操作过程。

3.5 MATLAB 的物理层模型

在本书中，我们会重复而系统地建立下行链路传输中必要的 LTE PHY 层组件。不过，因本书章节有限，我们会挑选重点详解。重复渐进设计在教学上比顺序讲解所有标准定义的参数和细节更有价值。

如本书题目，我们致力于通过扩充技术讨论及其 MATLAB 执行程序深入理解 LTE 标准。运行和执行系统程序和仿真加深了我们的理解层面。

最后，我们会重点介绍一些产品可以帮助用户用 MATLAB 建模、仿真、构建原型和执行无线系统。

3.6 MATLAB

MATLAB 是一种用途广泛的编程语言，用于算法开发、数据分析、可视化和数值运算。假如我们搜索所有提及它的技术期刊和出版物，就会发现 MATLAB 已经长期在通信系统设计方面为学术和工程界所使用。它可以使设计者更专注于算法而不是底层程序设计。它在无线系统建模方面有很多特点：

1）它有一个交互式程序和环境匹配科学的探索过程；

2）它对数据和算法的无缝存取；

3）它拥有很多可视化、算法开发和数据分析的工具。

矩阵作为基本数据类型： MATLAB 的基本数据类型是矩阵。因通信系统中大多数算法都是用块或帧处 理数据，这些算法可以很自然地植入 MATLAB。这意味着用矩阵表示的数学公式可以简单用 MATLAB 表示。比如，MIMO 系统中接受数据和发射数据之间的关系可以用系统线性方程 $y = Ax + n$ 表示。这类关系可以用几行 MATLAB 代码简单表示。这一算法如使用标准 C 代码表示，则需要两个 for 结构。

线性代数和傅里叶分析： MATLAB 包含了数学、统计，和工程函数以支持一般的工程和科学运算。这些函数，由数学专家们设计，是 MATLAB 语言的基础。核心数学函数使用了 LAPACK 和 BLAS 线性代数子程序库以及 FFTW 离散傅里叶变换库。线性代数、统计、傅里叶变换、滤波、优化和数值积分这些数学函数可在 MATLAB 中快速精确执行。

设计验证可视化： 大部分图形化需要的可视化工程科学数据都可在 MATLAB 中找到。它们包括 2D 和 3D 描点函数，3D 体可视化函数，互动描点以及将结果导出为所有流行图像格式。描点可以由用户通过多种方法定制。

复数和多种数据类型：通信系统仿真广泛使用复数和随机数。MATLAB 可以支持各种数据类型的算数运算，包括双精度、单精度和整型。MATLAB 还有随机数生成优化函数。函数如 randn（正态分布的随机数），rand（均匀分布），randi（离散整数随机分布），它们有周期性和效率方面的良好性能[1]。

3.7 MATLAB 工具箱

MATLAB 增值软件工具称为工具箱。它提供了包括如信号处理和通信的特殊数学函数。他们补充了核心 MATLAB 库，提供了专用函数和对象为建模和构建算法和系统加速。它们的算法性组件可使用户摆脱重复性的底层工作，专注顶层开发。

它的四个系统工具箱——DSP 系统工具箱[2]，通信系统工具箱[3]，相控阵列系统工具箱[4]和计算机视觉系统工具箱[5]——为不同应用领域的系统建模量身打造。它不仅仅为不同应用领域的设计、仿真和验证提供算法，它同时提供动态系统建模的测试平台。在最后一节，我们将会更详细回顾这些系统工具箱。

3.8 Simulink 组件

Simulink 组件是 MATLAB 下提供一个多维仿真和动态嵌入式系统建模设计的环境[6]。它提供了一个互动图形环境和定制化库设置。通过使用简单的图形化设计界面，Simulink 可帮助我们设计、仿真、执行和测试一系列实时系统，包括通信、控制、信号处理和视频处理。

通过 Simulink 和它简单易懂的预定义区块设置，可以创建、建模和维护我们系统的一个具体的模块。其他模块设置或系统工具箱也集成在 Simulink 中，它们包括特殊的功能，为空间技术、通信、射频、信号处理，视频、图像处理和其他应用提供设计支持。下面这些特性对通信系统的建模和仿真非常有用。

与 MATLAB 集成：MATLAB 函数可以被 Simulink 模型直接调用以执行这些算法分析数据和设计验证。在 simulink 中使用 MATLAB 函数模块可以将 MATLAB 代码集成进 Simulink。Simulink 将会首先使用代码生成工具转换 MATLAB 代码为 C 代码，随后编译 C 代码为 MEX（MATLAB 可执行文件）函数。它可以为 simulink 直接调用执行。

信号属性和数据类型支持：与 MATLAB 相似，Simulink 定义了以下信号和参数属性：数据类型——单精度，双精度，有符号或无符号的 8B，16B 或 32B 整型数；布尔型和定点型；维度——标量，向量，矩阵，或 N - D 数组；值——实数或复数。这可以让我们了解如实时有限字节长度在算法计算精度方面的影响。

仿真能力：在 Simulink 构建模型之后，我们可以仿真他的动态情况，观察结果。Simulink 提供了一些方法和工具确保仿真精度和速度，包括定步长和可变步长求解器，图形化调试器，和模型分析工具。

使用求解器：求解器是一些可以调用模型信息，计算系统实时动态特性的数值积分算法。Simulink 提供求解器支持连续时序（模拟），离散时序（数字），混合时序（混合信号）和多重速率系统的仿真。

执行仿真：当我们设定好模型的仿真操作后，我们可以通过使用 Simulink GUI（图形用户界面）或 MATLAB 命令行批处理模式，交互运行仿真。我们可以使用如下一些仿真模式：

1）一般（默认）模式，解释执行模型仿真；

2）加速模式，通过生成编译目标指令，加速仿真执行。该模式允许模型参数改变；

3）高速模式，通过从生成可执行片段在多核运行获得比加速模式更快的速度，但是解释性较少。

3.9 建模与仿真

大多数系统和组件的算法开发都是使用 MATLAB 开始的。通过数字信号处理、线性代数和数学运算库，算法设计在 MATLAB 中可以快速简单的分解成恰当的运算序列。独立算法可以设计和彼此相连，它们构成了系统模型的基础。MATLAB 和 Simulink 都可以很好地进行系统建模。如我们之前所述，Simulink 可集成 MATLAB 算法和函数生成系统组件。通过使用一系列增值工具箱，可以扩大系统范围仿真和验证它的工作是否符合定义。在本节中，我们会介绍一些 MATLAB 和 Simulink 工具箱以帮助我们的设计。

3.9.1 DSP 系统工具箱

DSP 系统工具箱提供了基础信号处理运算的算法和工具。它拥有强大的专用滤波器设计能力，FFT（快速傅里叶变换）和多速率处理能力。它捕捉系统对象的算法使处理流数据和建立实时系统原型的工作变得简单。DSP 系统工具箱拥有专用工具链接音频文件和设备，提供频谱分析，以及应用其他交互可视化技术分析系统动作和性能。所有的这一切组件都支持自动 C/C + +代码生成，大部分支持定点型数据，少部分可生成 HDL 代码。

3.9.2 通信系统工具箱

通信系统工具箱提供了算法和工具设计、仿真和分析通信系统。这个工具箱

特别为通信系统的 PHY 建模设计。它包括了含有源编码、信道编码、交织、调制、平衡、同步、MIMO 和信道建模的组件库。这些组件作为 MATLAB 函数、MATLAB 系统对象，Simulink 模块可在 MATLAB 和 Simulink 系统建模中使用。所有组件支持 C/C + + 代码生成，大部分支持定点型数据类型，少数支持生成 HDL 代码以在 FPGA 和 ASIC 上执行。

3.9.3 并行计算工具箱

并行计算工具箱[7]可以通过使用多核处理器，GPU（图像处理单元），计算机簇帮助加速计算，解决大数据密度的问题。通过并行 for 循环，特殊数组类型和并行数值算法使 MATLAB 应用程序并行处理。这个工具箱可在 Simulink 中使用并列运行多个模型仿真。两个加速仿真的主要过程如下表示：

多核或簇处理：很多程序可以通过分割为独立进程在不同处理器上同时执行来提速。这一类进程并行应用包括设计优化仿真，BER 测试和蒙特卡罗仿真。工具箱简单易用，提供了 parfor，一个并列 for 循环架构自动分散独立的进程到多个 MATLAB 处理工上。MATLAB 处理工是一个独立于 MATLAB 桌面运行的 MATLAB 计算引擎。MATLAB 可以自动检测到处理工在线，并在只有桌面在线的情况下启动处理工。进程执行也会通过其他方法启动，诸如工具箱中进程对象的操作。

GPU 处理：并行计算工具箱可以提供一个特殊数组类型允许有 CUDA 功能的 NVIDIA GPU 直接参与 MATLAB 计算。支持的功能有 FFT，元素智能运算和一系列线性代数运算。工具箱也提供机制允许现有的 CUDA GPU 核参与 MATLAB 运算。通信系统工具箱有很多特殊算法支持 GPU 处理。并行计算工具箱也可直接在 GPU 上运行多个通信算法。

3.9.4 定点型设计器

MATLAB 的定点型设计器[8]，即过去的定点型工具箱，支持定点型数据类型，操作和算法。使用定点型设计器，有限字长效应可以被若干个算法建模。工具箱支持通过使用 MATLAB 命令设计定点型算法，并可以和相同算法的浮点型结果做比较。这些算法可以在 Simulink 建模中调用导入或导出定点型数据。工具箱提供一组工具可以轻松把浮点型算法转换为定点型算法。

3.10 原型建模与实现

很多个 MathWorks 产品都可在 MATLAB 环境下帮助将设计从构想转化为嵌入式代码。MATLAB 算法必须首先基于设计规范进行优化为有限字长数据类型，

并受内存和复杂度的限制等。然后，它可集成到大系统模型并进行仿真。字节正确性检测可以验证 MATLAB 的软件和硬件执行结果是否最优参考。最后，生成 C 和 HDL 代码以进行硬件执行。MATLAB 维护简单的源设计代码以避免手动编程带来的错误。在本章中，我们将会介绍这些产品。

3.10.1 MATLAB 代码生成器

MATLAB 代码生成器[9] 可以从 MATLAB 代码生成完整的 C 和 C + + 代码。生成的源代码轻量可读。MATLAB 代码生成器可为 MATLAB 语言的大子集生成代码，包括程序控制结构、函数和矩阵运算。它也支持为函数和多个工具箱和系统工具箱的系统对象生成代码。MATLAB 代码生成器可以生成：

1） MEX 函数，可以使我们加速 MATLAB 代码中计算密集部分，验证代码的行为；

2） 可读和轻量级 C/C + + 代码，集成现有 C/C + + 源代码和环境；

3） 为集成基于 C 代码的工具和环境提供动态和静态库；

4） 为算法的原型建模和概念验证提供 C/C + + 可执行性。

3.10.2 硬件实现

通信系统设计既可以在嵌入式软件也可以在嵌入式硬件上实现。嵌入式软件在 DSP 和通用处理器上执行。从 MATLAB 模型到嵌入式软件执行的过程包括两个步骤：

1） C/C + + 代码生成；

2） 编译或硬编译 C 代码为汇编代码。

MATLAB 代码生成器可以完成第一步，而很多软件仿真工具的编译器可以为目标硬件完成第二部。

嵌入式软件执行在 FPGA 和 ASIC 上实现。从 MATLAB 模型到 FPGA 和 ASIC 执行的过程包括两个步骤：

1） HDL 代码生成器[10] 把 MATLAB 函数或 Simulink 模型转化为 VHDL 或 Verilog 代码。

2） 集成仿真环境预处理把 RTL（寄存器传输级） Verilog 和 VHDL 代码综合为 FPGA 或 ASIC 逻辑。它用来执行第一步的结果。

其他 MathWorks HDL 工具、HDL 验证器，使用 HDL 仿真器和 FPGA 硬件在环仿真器实现 Verilog 和 VHDL 设计验证自动化。HDL 验证器[11] 可以把 RTL 设计导入 MATLAB，通过比较相同算法在 MATLAB 和 Simulink 中仿真的结果验证 VHDL 和 Verilog 代码输出的正确性。因在本书中，我们关注 LTE 标准的建模、仿真和软件原型设计，HDL 代码硬件执行和设计实现不在我们的讨论范围。

3.11 系统对象介绍

在本书中，我们重点介绍通信系统工具箱的诸多特性，特别是介绍 MATLAB 中的新系统对象。在很多直观用户界面，系统对象可简化通信系统的构建，并使 MATLAB 代码更具可读性和复用性。系统对象可在 MATLAB 编程和 Simulink 建模中使用。它作为 MATLAB 对象表现有时效和可执行的算法，这些算法作为对象可以轻松使用并自我归档。鉴于在本书的以下部分中，我们基于 MATLAB 的系统对象进行 LTE 系统建模，本章中我们将先对如何使用算法组件进行一个简短的教学。

3.11.1 通信系统工具箱的系统对象

通信系统工具箱的系统对象属于通信（COMM）包，他的名字前缀以“comm”开头。调用所有通信系统工具包中的系统对象，只需在 MATLAB 命令栏内键入“comm.”，并按 Tab 键：

```
>> comm. <Tab>
```

随后就会在按字母排序的列表中出现工具箱内的所有系统对象。MATLAB 最新版的通信系统工具箱包含总共 123 个系统对象算法。

让我们选择一个系统对象，如 comm. QPSKModulator，建立一个此类型调制器的实例。我们管这个实例叫“Modulator”：

```
>> Modulator = comm. QPSKModulator
```

这个 QPSK（正交相移键控）调制器就会在 MATLAB 工作窗口显示生成并附带一个说明（见图 3. 1）。

Command Window

```
>> Modulator = comm.QPSKModulator

Modulator =

  System: comm.QPSKModulator

  Properties:
       PhaseOffset: 0.785398163397448
          BitInput: false
     SymbolMapping: 'Gray'
    OutputDataType: 'double'
```

图 3. 1　从通信系统工具箱创建一个系统对象

每个系统对象包含属性和方法。创建对象时会显示默认属性；这个自组织文档是系统对象的一个有用功能。通过观察一个系统对象的属性列表，我们可以知道什么参数和值可以更改。比如，QPSK 的相移默认是 ~/4. 我们可以更改这个值为 ~/2。我们可以通过两种途径更新属性：

1）新建一个对象后通

过" ." 标记改变默认属性。如：

```
>> Modulator = comm.QPSKModulator;
>> Modulator.PhaseOffset = pi/2
```

2）在新建对象时设置属性。如：

```
>> Modulator = comm.QPSKModulator ('PhaseOffset', pi/2);
```

如属性为字符串或字符，会显示一个可能属性列表，方便我们选择想要的属性进行更改。比如，当我们键入" Modulator. SymbolMapping = " 然后敲一下 Tab 键，就会出现一个可用属性值设置选项列表，里面有" Binary" 和 “Gray” 两个选项。

单步法是执行系统对象的主要方法。当一个系统对象创建并设置好后，单步法代入一个输入（或多个输入）并得到一个输出（或多个输出）。两种命令可以用单步发执行系统对象，我们可以：

1）通过“.”标记符调用系统对象：y = Modulator. step（u）。

2）通过单步法函数，将系统对象作为函数的声明：y = step（Modulator，u）。

图 3.2 中，用 MATLAB randi 函数创建一个 10×1 列字节向量（变量 u），并将其代入 Modulator 系统对象。通过调用单步方法，可根据对象属性得到一个带 QPSK 调制的 5×1 输出向量（y）。

现在我们知道了如何访问、创建、设置属性、配置和调用系统对象进行运算，我们下面将会用系统对象建一个简单的通信系统。

```
>> u=randi([0 1], 10, 1)

u =

     1
     1
     0
     1
     1
     0
     0
     1
     1
     1

>> y=step(Modulator,u);
>> y=Modulator.step(u)

y =

   0.7071 - 0.7071i
  -0.7071 + 0.7071i
  -0.7071 - 0.7071i
  -0.7071 + 0.7071i
   0.7071 - 0.7071i
```

图 3.2　调用单步方法执行系统对象

3.11.2　系统对象的测试平台

下面这段 MATLAB 脚本，或者叫测试平台，使用系统对象对一个简单收发系统进行 BER 分析。收发器由一个 QPSK 调制器、加性白噪声（AWGN）信道和一个 QPSK 解调器构成。注意代码中包括

4 个通信系统工具箱的系统对象：comm. QPSKModulator， comm. AWGNChannel， comm. QPSKDemodulator 和 comm. ErrorRate。

Algorithm

MATLAB script

```
%% Constants
FRM=2048;
MaxNumErrs=200;MaxNumBits=1e7;
EbNo_vector=0:10;BER_vector=zeros(size(EbNo_vector));
%% Initializations
Modulator      = comm.QPSKModulator('BitInput',true);
AWGN             = comm.AWGNChannel;
DeModulator =  comm.QPSKDemodulator('BitOutput',true);
BitError          = comm.ErrorRate;
%% Outer Loop computing Bit-error rate as a function of EbNo
for EbNo = EbNo_vector
   snr = EbNo + 10*log10(2);
   AWGN.EbNo=snr;
   numErrs = 0; numBits = 0;results=zeros(3,1);
   %% Inner loop modeling transmitter, channel model and receiver for each EbNo
   while ((numErrs < MaxNumErrs) && (numBits < MaxNumBits))
      % Transmitter
      u            = randi([0 1], FRM,1);                  % Generate random bits
      mod_sig = step(Modulator,   u);                % QPSK Modulator
      % Channel
      rx_sig  = step(AWGN,       mod_sig);          % AWGN channel
      % Receiver
      y =       step(DeModulator, rx_sig);              % QPSK Demodulator
      results = step(BitError,    u, y);                 % Update BER
      numErrs = results(2);
      numBits = results(3);
   end
   % Compute BER
   ber = results(1); bits= results(3);
   %% Clean up & collect results
   reset(BitError);
   BER_vector(EbNo+1)=ber;
end
%% Visualize results
EbNoLin = 10.^(EbNo_vector/10);
theoretical_results = 0.5*erfc(sqrt(EbNoLin));
semilogy(EbNo_vector, BER_vector)
grid;title('BER vs. EbNo - QPSK modulation');
xlabel('Eb/No (dB)');ylabel('BER');hold;
semilogy(EbNo_vector,theoretical_results,'dr');hold;
legend('Simulation','Theoretical');
```

这个脚本由 4 部分构成。在初始化部分，程序创建系统对象并设置一些参数。第二部分包括迭代 Eb/N0 值和计算相应的 BER。第三部分为收发处理循环，程序调用单步方法调制输入信号、添加信道噪声，解调得到接收信号并计算 BER。最后，在第四部分，完成仿真并输出 BER 可视化结果。

图 3.3 对比了仿真结果和理论分析结果

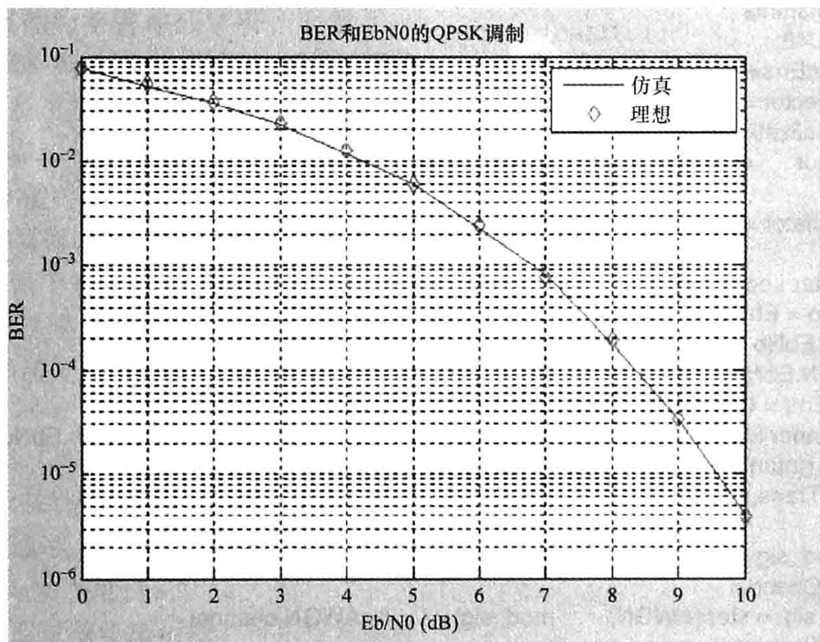


图 3.3 AWGN 信道内 QPSK 调制的 BER 曲线；仿真和理论结果一致

通过这个脚本，我们得到了使用 AWGN 信道 QPSK 调制系统的 BER 曲线。理论上这个系统的 BER 可由下式表示：

$$\mathrm{BER} = \frac{1}{2}\mathrm{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{3.1}$$

使用系统对象使 MATLAB 代码模块化并简单易懂，它可以构建更复杂的系统。我们在本书中将会按照下面这 4 个步骤进行仿真：初始化、处理循环、结束、输出可视化结果。提高 MATLAB 编程和可读性的办法是分割测试脚本，和算法与系统描述的可视化操作。下面我们将会展示如何通过为我们的收发器调用 MATLAB 函数，以及分割测试脚本中的算法进一步达到这一目标。

3.11.3 系统对象函数

MATLAB 函数 chap3_ ex02_ qpsk 为我们这个简单的 QPSK 收发系统的算法部分。这个函数包括三个变量：

1）第一个输入是一个标量，对应 Eb/N0；

2）第二个变量是一个停止边界条件，为仿真停止之前可观察的最大错误字符数；

3）第三个变量是另一个停止边界条件，为仿真停止之前可处理的最大字符数。

程序对每个 Eb/N0 值运行 While 环，直到满足最大可处理字符数和最大可观察错误数的条件满足。程序通过调用单步方法执行每个系统对象。它可得到两个结果：

1）BER，定义为错误比特数与总处理比特数之比；

2）总处理比特数，与第二和第三输入变量定义的停止条件有关。

Algorithm

MATLAB function

```
function [ber, bits]=chap3_ex02_qpsk(EbNo, maxNumErrs, maxNumBits)
%% Initializations
persistent Modulator AWGN DeModulator BitError
if isempty(Modulator)
   Modulator     = comm.QPSKModulator('BitInput',true);
   AWGN          = comm.AWGNChannel;
   DeModulator = comm.QPSKDemodulator('BitOutput',true);
   BitError      = comm.ErrorRate;
end
%% Constants
FRM=2048;
M=4; k=log2(M);
snr = EbNo + 10*log10(k);
AWGN.EbNo=snr;
%% Processsing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0;results=zeros(3,1);
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u            = randi([0 1], FRM,1);              % Random bits generator
   mod_sig    = Modulator.step(u);              % QPSK Modulator
   % Channel
   rx_sig      = AWGN.step(mod_sig);             % AWGN channel
   % Receiver
   demod      = DeModulator.step(rx_sig);     % QPSK Demodulator
   y            = demod(1:FRM);                     % Compute output bits
   results     = BitError.step(u, y);               % Update BER
   numErrs    = results(2);
   numBits    = results(3);
end
%% Clean up & collect results
ber = results(1); bits= results(3);
reset(BitError);
```

为了避免每次我们调用函数时创建和释放系统对象的开销，函数内的系统对象为持久变量声明类型。持久变量声明类型可以支持只在函数第一次调用时创建系统对象。这将提供函数调用的效率，从而提高我们循环调用函数时的仿真速度。

3.11.4　字符误码率仿真

通信系统工具箱提供 BERTool 作为 BER 仿真的集成测试平台。BERTool 是一个图形化应用，可以计算一系列字符误码率仿真并与理论值进行比较。

举例说明，函数 chap3_ ex02_ qpsk. m 的 BER 仿真可视化，如图 3.4 所示，进入 Monte Carlo 选项卡，选择仿真 MATLAB 文件，选择 Eb/N0 值和停止条件。BERTool 将会对范围内的 Eb/N0 计算 BER 并自动显示结果。通过进入 Theoretical 选项卡选择使用的调制和编码方案，可以比较仿真结果可以与理论值。图 3.5 所示为一个 BERTool 比较结果的例子。

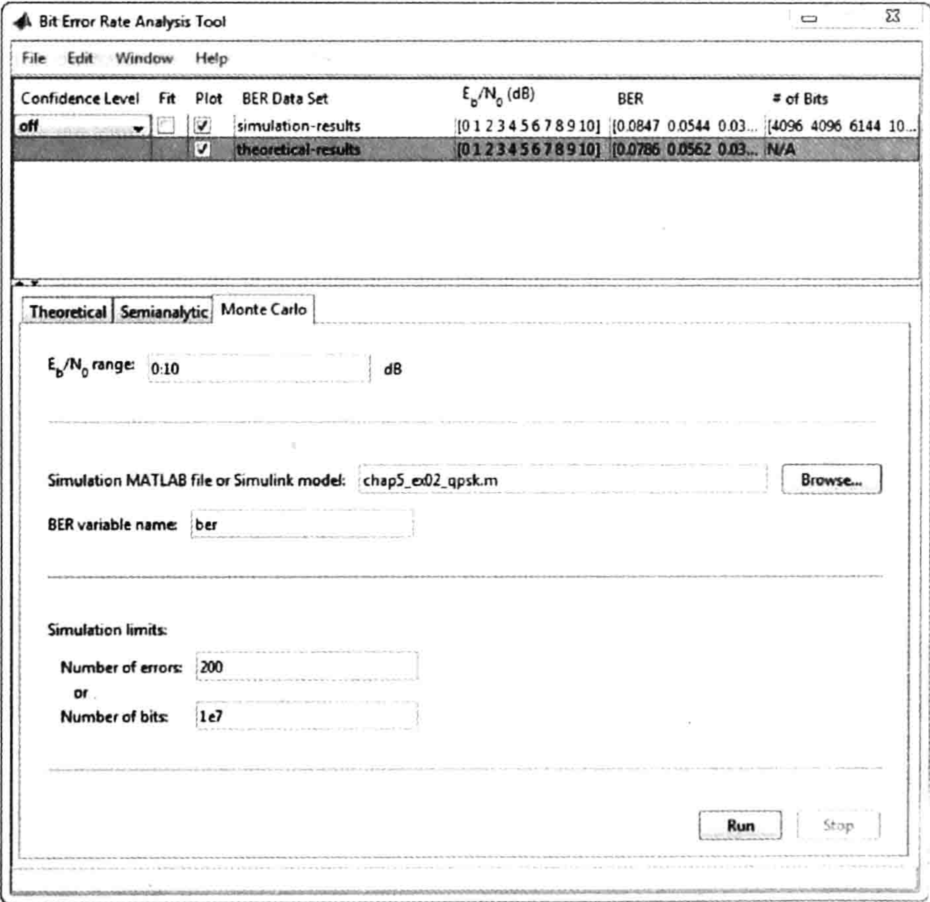


图 3.4　BERTool：BER 结果可视化的测试应用程序

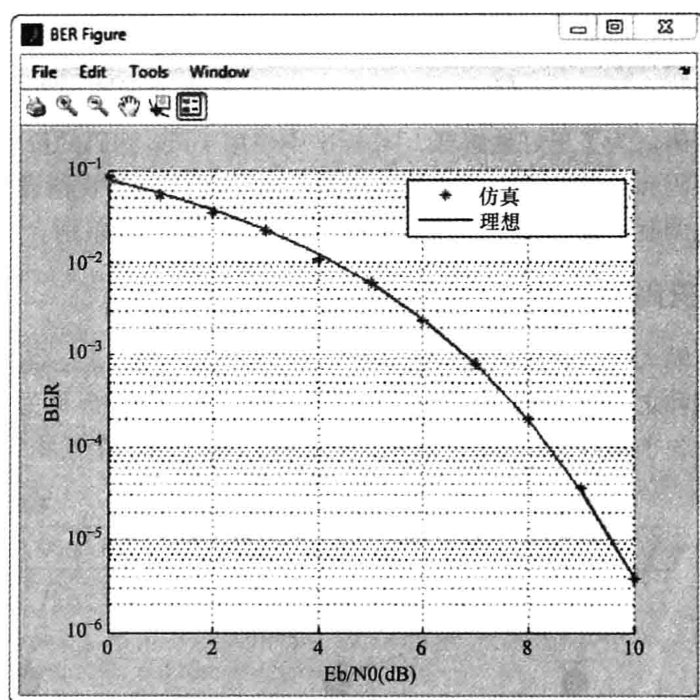


图 3.5 BER 仿真和理论值比较：QPSK 和 AWGN 信道

3.12 MATLAB 信道编码实例

在本节中，通过一系列简单易懂的 MATLAB 程序，我们将会研究如何使用 Toolbox 进行进到编码。首先，我们建立使用一个使用卷积编码和硬判决译码的 Viterbi 译码器的系统模型。然后我们会更新算法，使用软判决译码。最后，我们会使用 Turbo 编码算法替代卷积编码，并比较每一步的性能。通过这个简单的练习，我们不仅学习如何简单地使用 MATLAB 和通信系统工具箱提升移动通信系统建模的复杂度，而且我们也将会很清晰地看到 Turbo 编码在 LTE 标准中显著改善 BER 性能。

3.12.1 纠错与检错

信道编码包含检错和纠错功能。通过使用 CRC（循环冗余检查）检错器检错，接收器可以请求和重传，即自动请求重传功能。前向错误更正编码可以通过包含传输信号的冗余字节纠错。集成检错和前向错误更正的功能即 HARQ（混合

型自动请求同传），它在大多数 3G 协议中，也在 LTE 标准中使用。纠错编码一般分为块编码和卷积编码。卷积编码广泛应用于 2G 和 3G 移动通信标准中，主要因为它复杂度较小。

在本节中，我们会详细说明 MATLAB 模型，它现在已经包括了调制，下面将加入信道编码。为了更好地解释 LTE 标准中使用 Turbo 编码的价值和动机，我们会比较卷积码和 Turbo 码性能。不仅如此，为了解释在接收器设计中的平衡，我们会比较在调制 - 编码中使用和不使用软判决译码的性能差别。

3.12.2 卷积码

卷积码由输入序列和编码器冲击响应的卷积生成。编码器接收 $k$bit 输入，并通过当前数据和之前 m 个输入，生成 $n$bit 输出。编码器的编码率为 $R_c = k/n$，故卷积编码器由 3 个参数（n，k，m）确定。图 3.6 所示为一个卷积编码器。

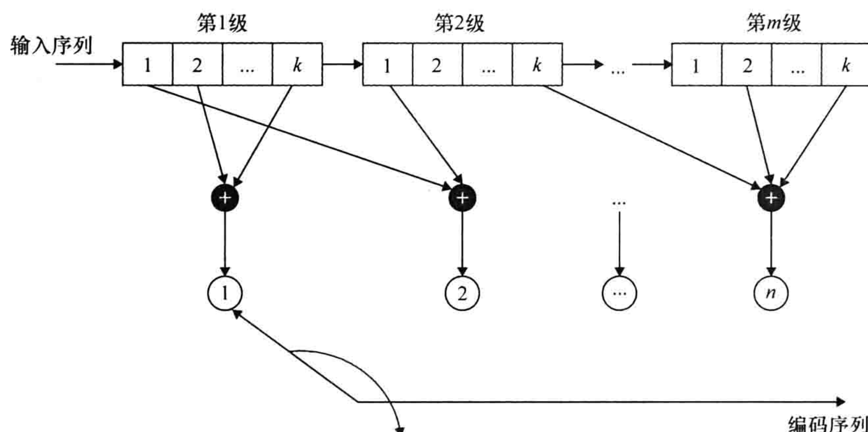


图 3.6 卷积编码器（n，k，m）的结构

3.12.3 硬判决 Viterbi 译码

在这个练习中，我们首先更新上一节中的 MATLAB 函数，在调制中添加信道编码方案。当使用信道编码方案时，发射端通过无线信道发送包括信息字节的冗余字节。接收端接收信号并利用冗余字节检测并纠错信道传输中的误码。让我们开始向这个通信系统中添加卷积编码器和 Viterbi 译码器。通信系统使用硬判决 Viterbi 译码器，解调器把接收信号映射为字节串并将其送至 Viterbi 译码器纠错。下面这个 MATLAB 函数（chap_ ex03_ qpsk_ viterbi）使用 QPSK 调制和硬判决 Viterbi 译码附加 AWGN 信道条件。

Algorithm

MATLAB function

```
function [ber, bits]=chap3_ex03_qpsk_viterbi(EbNo, maxNumErrs, maxNumBits)
%% Initializations
persistent Modulator AWGN DeModulator BitError ConvEncoder Viterbi
if isempty(Modulator)
   Modulator     = comm.QPSKModulator('BitInput',true);
   AWGN          = comm.AWGNChannel;
   DeModulator =  comm.QPSKDemodulator('BitOutput',true);
   BitError         = comm.ErrorRate;
   ConvEncoder=comm.ConvolutionalEncoder(...
      'TerminationMethod','Terminated');
   Viterbi=comm.ViterbiDecoder('InputFormat','Hard',...
      'TerminationMethod','Terminated');
end
%% Constants
FRM=2048;
M=4; k=log2(M); codeRate=1/2;
snr = EbNo + 10*log10(k) + 10*log10(codeRate);
AWGN.EbNo=snr;
%% Processsing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0;results=zeros(3,1);
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u              = randi([0 1], FRM,1);                 % Random bits generator
   encoded    = ConvEncoder.step(u);               % Convolutional encoder
   mod_sig    = Modulator.step(encoded);        % QPSK Modulator
   % Channel
   rx_sig       =  AWGN.step(mod_sig);             % AWGN channel
   % Receiver
   demod       =  DeModulator.step(rx_sig);       % QPSK Demodulator
   decoded     = Viterbi.step(demod);                % Viterbi decoder
   y               = decoded(1:FRM);                        % Compute output bits
   results       = BitError.step(u, y);                   % Update BER
   numErrs     = results(2);
   numBits      = results(3);
end
%% Clean up & collect results
ber = results(1); bits= results(3);
reset(BitError);
```

通过运行这个函数和 BERTool，我们可以判断硬判决 Viterbi 译码器的性能，并比较理论上限结果。通过观察图 3.7 中的结果，可以看到 BER 仿真曲线低于理论上限值，如我们预期。这个结果表明我们需要改进我们的译码算法以达到更好的性能。

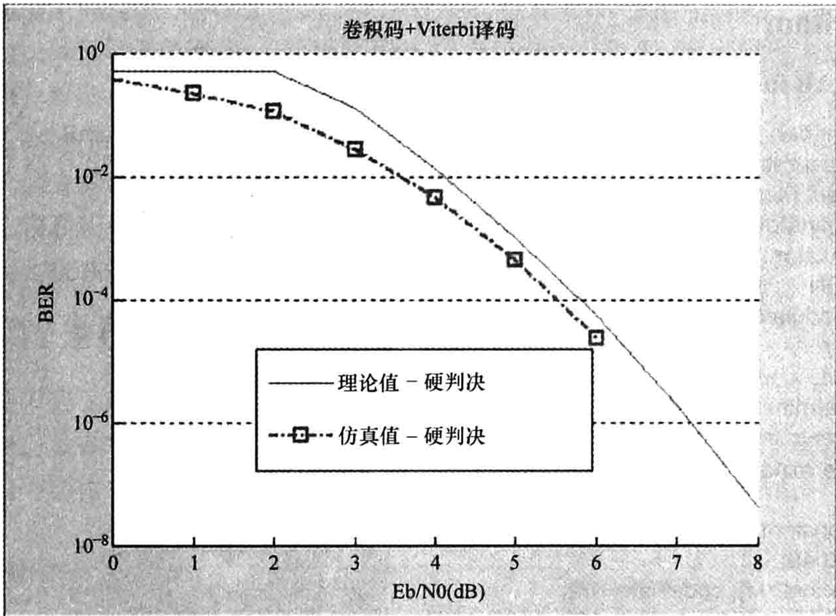


图3.7 硬判决Viterbi译码器性能：QPSK调制附加AWGN信道条件

3.12.4 软判决Viterbi译码

我们将会通过使用软判决译码算法改进BER性能。在软判决译码中，解调器映射接收信号为对数似然比。这个概率测量基于接收到正确信号而不是错误信号的对数似然值。当对数似然比作为Viterbi译码器输入时，BER性能可得到提升。这个算法可以通过改变一小部分解调器和Viterbi译码器系统对象参数实现。下面即为使用软判决Viterbi译码器的MATLAB函数（chap3_ ex04_ qpsk_ viterbi_ soft）。

Algorithm

MATLAB function

```
function [ber, bits]=chap3_ex04_qpsk_viterbi_soft(EbNo, maxNumErrs, maxNumBits)
%% Initializations
persistent Modulator AWGN DeModulator BitError ConvEncoder Viterbi Quantizer
if isempty(Modulator)
   Modulator      = comm.QPSKModulator('BitInput',true);
   AWGN           = comm.AWGNChannel;
   DeModulator =  comm.QPSKDemodulator('BitOutput',true,...
      'DecisionMethod','Log-likelihood ratio',...
      'VarianceSource', 'Input port');
   BitError       = comm.ErrorRate;
   ConvEncoder=comm.ConvolutionalEncoder(...
      'TerminationMethod','Terminated');
   Viterbi=comm.ViterbiDecoder(...
```

```
        'InputFormat','Soft',...
        'SoftInputWordLength', 4,...
        'OutputDataType', 'double',...
        'TerminationMethod','Terminated');
    Quantizer=dsp.ScalarQuantizerEncoder(...
        'Partitioning', 'Unbounded',...
        'BoundaryPoints', -7:7,...
        'OutputIndexDataType','uint8');
end
%% Constants
FRM=2048;
M=4; k=log2(M); codeRate=1/2;
snr = EbNo + 10*log10(k) + 10*log10(codeRate);
noise_var = 10.^(-snr/10);
AWGN.EbNo=snr;
%% Processsing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0;results=zeros(3,1);
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
    % Transmitter
    u           = randi([0 1], FRM,1);              % Random bits generator
    encoded     = ConvEncoder.step(u);              % Convolutional encoder
    mod_sig     = Modulator.step(encoded);          % QPSK Modulator
    % Channel
    rx_sig      = AWGN.step(mod_sig);               % AWGN channel
    % Receiver
    demod       = DeModulator.step(rx_sig, noise_var);  % Soft-decision QPSK Demodulator
    llr         = Quantizer.step(-demod);           % Quantize Log-Likelihood Ratios
    decoded     = Viterbi.step(llr);                % Viterbi decoder with LLRs
    y           = decoded(1:FRM);                   % Compute output bits
    results     = BitError.step(u, y);              % Update BER
    numErrs     = results(2);
    numBits     = results(3);
end
%% Clean up & collect results
ber = results(1); bits= results(3);
reset(BitError);
```

理论上讲，我们希望得到 2dB 的性能提升，如图 3.8 中所示的实际仿真结果。下面我们将会考察使用 Turbo 编码能否得到更好的 BER 结果。

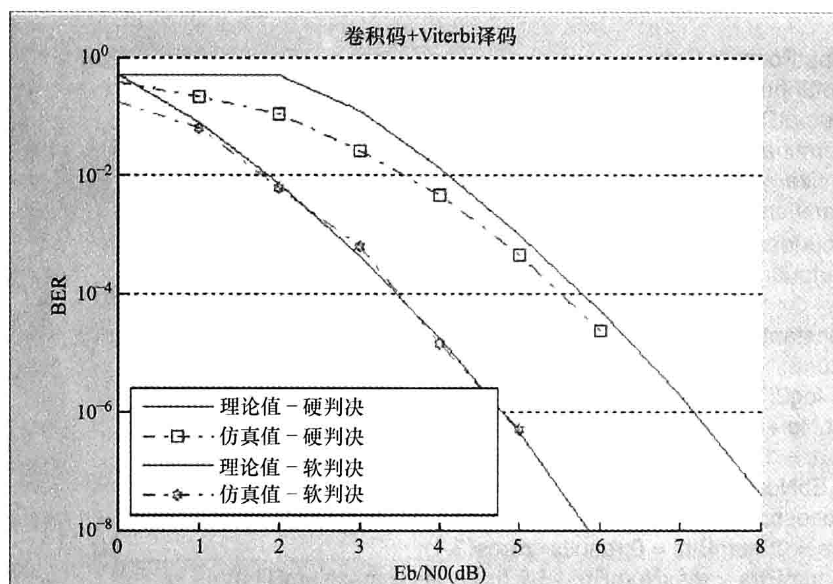


图 3.8　软判决 Viterbi 译码器性能：QPSK 调制附加 AWGN 信道条件

3.12.5　Turbo 编码

Turbo 编码超过软判决 Viterbi 译码显著提升 BER 性能。Turbo 编码在发射端并行使用两个卷积编码器，在接收端串行使用 2 个后验概率译码器（APP）。本例使用 1/3 码率的 Turbo 编码器。对每个输入字节，输出包括一个系统字节和 2 个奇偶校验位，共 3bit 输出。

下面这个 MATLAB 函数使用 Turbo 编码器和译码器。注意译码器使用了一个重复迭代，其性能将会随着迭代次数增加而提升。在本例中，我们在译码器中进行 6 次迭代。

Algorithm

MATLAB function

```
function [ber, bits]=chap5_ex05_qpsk_turbo(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2048;
Trellis=poly2trellis(4, [13 15], 13);
Indices=randperm(FRM);
M=4;k=log2(M);
R= FRM/(3* FRM + 4*3);
snr = EbNo + 10*log10(k) + 10*log10(R);
noise_var = 10.^(-snr/10);
%% Initializations
```

```
persistent Modulator AWGN DeModulator BitError TurboEncoder TurboDecoder
if isempty(Modulator)
   Modulator     = comm.QPSKModulator('BitInput',true);
   AWGN          = comm.AWGNChannel;
   DeModulator =  comm.QPSKDemodulator('BitOutput',true,...
      'DecisionMethod','Log-likelihood ratio',...
      'VarianceSource', 'Input port');
   BitError       = comm.ErrorRate;
   TurboEncoder=comm.TurboEncoder(...
      'TrellisStructure',Trellis,...
      'InterleaverIndices',Indices);
   TurboDecoder=comm.TurboDecoder(...
      'TrellisStructure',Trellis,...
      'InterleaverIndices',Indices,...
      'NumIterations',6);
end
%% Processsing loop modeling transmitter, channel model and receiver
AWGN.EbNo=snr;
numErrs = 0; numBits = 0;results=zeros(3,1);
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u              = randi([0 1], FRM,1);                        % Random bits generator
   encoded     = TurboEncoder.step(u);                          % Turbo Encoder
   mod_sig     = Modulator.step(encoded);                       % QPSK Modulator
   % Channel
   rx_sig        =  AWGN.step(mod_sig);                          % AWGN channel
   % Receiver
  demod       = DeModulator.step(rx_sig, noise_var);    % Soft-decision QPSK Demodulator
   decoded     = TurboDecoder.step(-demod);                      % Turbo Decoder
   y              = decoded(1:FRM);                              % Compute output bits
   results      = BitError.step(u, y);                          % Update BER
   numErrs     = results(2);
   numBits      = results(3);
end
%% Clean up & collect results
ber = results(1); bits= results(3);
reset(BitError);
```

图 3.9 所示为 AWGN 信道下 QPSK 调制 Turbo 编码的仿真结果。在 Eb/N0 为 1dB 的位置，我们可以看到 BER 值与硬判决 5dB 处值相同，与软判决 3dB 处值相同。这个结果明显反映了 Turbo 编码算法的优势。注意性能的提高会伴随着可计算复杂性的增大。如 Turbo 译码器通过了 6 次迭代才获得上述性能。我们将会在下一章中学习 turbo 译码器性能与复杂度的折中。

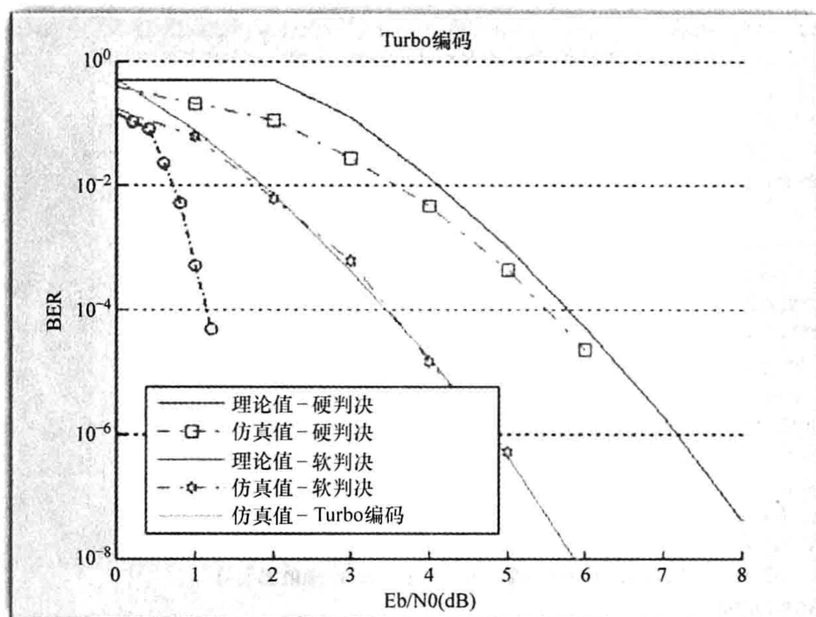


图 3.9 Turbo 编码器性能：QPSK 调制附加 AWGN 信道条件

3.13 本章小结

MATLAB，Simulink 和它们的工具箱为我们提供了强大的建模、仿真，评估性能，并最后生成并执行通信系统代码。我们可以使用从通信系统工具箱中调用算法构建模块，如系统对象和 Simulink 模块，进行建模和仿真。许多移动通信系统的 PHY 处理过程都可以在 MATLAB 中高效的建模和仿真。我们可以关注在系统模型中引入更先进的功能，而并不需要关注如何创建组件模块本身，如调制器和编码器。我们特别关注通信系统工具箱的系统对象。系统对象是一个可以自归档、使用方便，用户定制的建模和仿真组件，适用模块化流水线系统的建模。

当仿真复杂系统时，需要多个加速方法。这些方法帮助我们在可接受仿真时间内处理大数据和得到正确统计评估。MATLAB 工具包如并行处理工具包和 MATLAB 代码生成器可以加速仿真。最后，为了软件或硬件执行我们的设计，可以使用代码生成产品如自动 C 或 HDL 代码生成器，而得到我们想要的精确执行代码。

参考文献

[1] MathWorks Documentation Center, http://www.mathworks.com/help/MATLAB/random-number-generation.html (accessed 16 August 2013).
[2] MathWorks DSP System Toolbox, http://www.mathworks.com/products/dsp-system (accessed 16 August 2013).
[3] MathWorks Communications System Toolbox, http://www.mathworks.com/products/communications (accessed 16 August 2013).
[4] MathWorks Phased Array System Toolbox, http://www.mathworks.com/products/phased-array (accessed 16 August 2013).
[5] MathWorks Computer Vision System Toolbox, http://www.mathworks.com/products/computer-vision (accessed 16 August 2013).
[6] MathWorks Simulink, http://www.mathworks.com/products/simulink (accessed 16 August 2013).
[7] MathWorks Parallel Computing Toolbox, http://www.mathworks.com/products/parallel-computing (accessed 16 August 2013).
[8] MathWorks Fixed-Point Designer, http://www.mathworks.com/products/fixed-point-designer (accessed 16 August 2013).
[9] MathWorks MATLAB Coder, http://www.mathworks.com/help/coder/index.html (accessed 16 August 2013).
[10] MathWorks HDL Coder, http://www.mathworks.com/products/hdl-coder (accessed 16 August 2013).
[11] MathWorks HDL Verifier, http://www.mathworks.com/products/hdl-verifier (accessed 16 August 2013).

4 调制和编码

LTE（长期演进）下行链路 PHY（物理）层处理链可以认为是下行链路公共信道（DLSCH）和物理下行链路公共信道（PDSCH）处理的组合。DLSCH 即下行链路传输信道（TrCH）。它包括循环冗余检查（CRC）码添加、数据子块处理、信道 Turbo 编码、速率匹配、混合自动重传请求和码字重组这几个步骤。码字是 PDSCH 处理的输入，该处理包括绕码、调制、多天线多输入多输出（MI-MO）、时 - 频资源映射和正交频分复用（OFDM）传输。我们在下面三章内，分别从 DLSCH 和 PDSCH 处理这两个步骤入手，将他们分为 3 个部分讨论。

在本章中，我们考察 LTE 标准中的调制和编码方案。它们包括所有 DLSCH 和 PDSCH 处理步骤，但不包括 MIMO 和 OFDM 操作。接着，我们会考察 TrCH 处理，比较一系列映射逻辑信道和用户数据到码字的操作，而码字是共享物理信道的输入。

我们会建立 MATLAB 程序在发射端和接受端完整定义 TrCH 处理。我们也将用 MATLAB 函数，研究不同调制方案和不同编码率对加性白高斯噪声信道模型下比特误码率（BER）性能的影响。这些操作完整定义了用户数据比特如何处理和生成输入符号，然后通过 MIMO 和 OFDM 函数模块序列发送的。有关 MIMO 和 OFDM 的细节将会在后两章讨论考察。

4.1 LTE 调制方案

LTE 使用的调制方案包括 QPSK（正交相移键控）、16QAM（正交幅度调制），和 64QAM。图 4.1 所示为三种调制方案的星座图。

QPSK 调制情况下，每个调制符号各不相同，映射星座图上四个不同的位置。QPSK 需要 2 比特编码这 4 个里每一个不同的调制符号。16QAM 调制使用 16 个不同的信号选择，因此使用 4 个 bit 信息编码每一个调制符号。64QAM 调制有 64 个不同的可能值，因此需要 6bit 反映一个信号调制符号。

这几种调制方案，根据信道条件不同，在适应性调制中使用。当无线电链路相对清晰（就是说信噪比（SNR）较高），我们可以用较密的星座图调制方案，如 64QAM。在这样的情况下，发送一个信号符号需要 6bit，因此增加了信号的吞吐量。不过，当信道噪声水平增加，我们就要依靠使用码间隔离性更好的调制方案，如 QPSK。这会减少每个采样信号的比特数以及减小吞吐量。

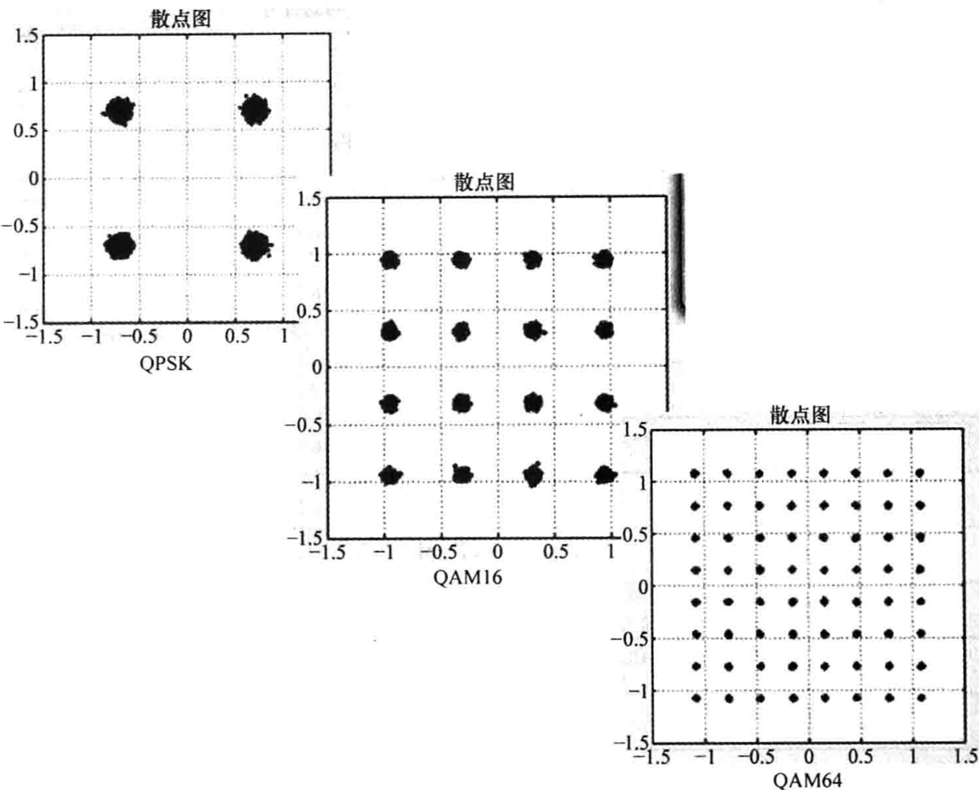


图 4.1 QPSK、16QAM，和 64QAM 的星座图

LTE 调制映射器，定义了如何分配调制符号到每个比特序列。表 4.1 为 QPSK，表 4.2 为 16QAM 调制器。由于篇幅有限，我们推荐读者参阅参考文献 [1] 了解 64QAM 调制映射的标准文件。

表 4.1 LTE QPSK 调制器映射

载荷比特图形/2bit	调制符号	
	同相（I）	正交（Q）
00	$1/\sqrt{2}$	$1/\sqrt{2}$
01	$1/\sqrt{2}$	$-1/\sqrt{2}$
10	$-1/\sqrt{2}$	$1/\sqrt{2}$
11	$-1/\sqrt{2}$	$-1/\sqrt{2}$

表 4.2 LTE 16QAM 调制器映射

载荷比特图形/4bit	调制符号	
	同相（I）	正交（Q）
0000	$1/\sqrt{10}$	$1/\sqrt{10}$
0001	$1/\sqrt{10}$	$3/\sqrt{10}$
0010	$3/\sqrt{10}$	$1/\sqrt{10}$
0011	$3/\sqrt{10}$	$3/\sqrt{10}$
0100	$1/\sqrt{10}$	$-1/\sqrt{10}$
0101	$1/\sqrt{10}$	$-3/\sqrt{10}$
0110	$3/\sqrt{10}$	$-1/\sqrt{10}$
0111	$3/\sqrt{10}$	$-3/\sqrt{10}$
1000	$-1/\sqrt{10}$	$1/\sqrt{10}$
1001	$-1/\sqrt{10}$	$3/\sqrt{10}$
1010	$-3/\sqrt{10}$	$1/\sqrt{10}$
1011	$-3/\sqrt{10}$	$3/\sqrt{10}$
1100	$-1/\sqrt{10}$	$-1/\sqrt{10}$
1101	$-1/\sqrt{10}$	$-3/\sqrt{10}$
1110	$-3/\sqrt{10}$	$-1/\sqrt{10}$
1111	$-3/\sqrt{10}$	$-3/\sqrt{10}$

我们注意比特到符号的映射既不基于一般的二进制又不基于格雷码方案。而且，LTE 定义了一种习惯性星座映射。LTE 也定义调制符号的平均信号功率在一定程度上归一化。

4.1.1 MATLAB 实例

作为 LTE 下行链路处理建模的第一步，我们从 LTE 调制方案开始。下面两个 MATLAB 函数表示了如何简单部署 LTE 调制和解调算法，包括所有定义，使用通信系统工具箱的系统对象。

Algorithm

MATLAB function

```
function y=Modulator(u, Mode)
%% Initialization
persistent QPSK QAM16 QAM64
if isempty(QPSK)
   QPSK      = comm.PSKModulator(4, 'BitInput', true, ...
                'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
                'CustomSymbolMapping', [0 2 3 1]);
   QAM16     = comm.RectangularQAMModulator(16, 'BitInput',true,...
                 'NormalizationMethod','Average power',
                  'SymbolMapping', 'Custom', ...
                  'CustomSymbolMapping',
                  [11 10 14 15 9 8 12 13 1 0 4 5 3 2 6 7]);
   QAM64     = comm.RectangularQAMModulator(64, 'BitInput',true,...
                 'NormalizationMethod','Average power',
                 'SymbolMapping', 'Custom',
                  'CustomSymbolMapping',
                  [47 46 42 43 59 58 62 63 45 44 40 41 ...
                  57 56 60 61 37 36 32 33 49 48 52 53 39
                  38  34 35 51 50 54 55 7 6 2 3 19 18 22 23 5
                  4 0 1 17 16 20 21 13 12 8 9 25 24 28 29 15
                  14 10 11 27 26 30 31]);
end
%% Processing
switch Mode
   case 1
      y=step(QPSK, u);
   case 2
      y=step(QAM16, u);
   case 3
      y=step(QAM64, u);
end
```

这个调制器函数有两个输入变量声明：输入比特流（u）和反应调制模式的参数（Mode）。函数计算出调制符号作为输出。函数执行 LTE 标准中使用的三种调制方案。如在 QPSK 情况下，我们用一个 comm. PSKModulator 系统对象并将其调制阶数设为 4。与此相似，在 16QAM 和 64QAM 中，我们用 comm. RectangularQAMModulator 系统对象并分别设置其调制阶数为 16 和 64。根据调制模式值的不同，处理输入比特生成调制符号作为输出。

为了保证系统对象描述 LTE 标准的精确性，我们还可以设置其他的属性：

1）可以设置 BitInput = true。这表示将调制器输入设为比特值向量。比如，QPSK 调制器下，因每 2bit 被映射到一个调制符号，输出向量的大小为输入向量

的一半。

2）可以设置 PhaseOffset = pi/4。这表示调制符号对应复平面单位圆上 4 个点，它们的角度依次为：[3π/4，π/4，-π/4，-3π/4]。

3）使用 CustomSymbolMapping 属性，我们可以确保 LTE 中定义的比特图形产生相应的输出符号。

注意，LTE 标准如同其他移动标准一样，没有任何对接收端的推荐配置。因此，本书中所有涉及接收端的定义都可以认为是对发射端操作的单纯反向。这其中反向操作最好的反映了复原传输比特估计的工作。即使没有在标准中有所定义，接收端反向操作以评估系统精度和性能的部分也是有必要的。

因解调是调制的反向操作，我们现在可以进行一些典型的解调设置。在解调器函数中，可同样用三种 LTE 中的调制类型，对应于调制模式，处理输入符号生成解调输出。如同前面章节中所讨论的，解调可以使用硬判决解码或软判决解码。当使用硬判决解码时，解调器输入符号映射到估计比特；而软判决解码时，输出为对数似然估计比（LLR）向量。函数 DemodulatorHard. m 为使用硬判决解码的解调。函数代入接收端调制符号（u）和调制模式（Mode）为输入。函数输出为解调比特。

Algorithm

MATLAB function

```
function y=DemodulatorHard(u, Mode)
%% Initialization
persistent QPSK QAM16 QAM64
if isempty(QPSK)
   QPSK = comm.PSKDemodulator(...
      'ModulationOrder', 4, ...
      'BitOutput', true, ...
      'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
      'CustomSymbolMapping', [0 2 3 1]);

   QAM16 = comm.RectangularQAMDemodulator(...
      'ModulationOrder', 16, ...
      'BitOutput', true, ...
      'NormalizationMethod', 'Average power', 'SymbolMapping', 'Custom', ...
      'CustomSymbolMapping', [11 10 14 15 9 8 12 13 1 0 4 5 3 2 6 7]);

   QAM64 = comm.RectangularQAMDemodulator(...
   'ModulationOrder', 64, ...
   'BitOutput', true, ...
   'NormalizationMethod', 'Average power', 'SymbolMapping', 'Custom', ...
   'CustomSymbolMapping', ...
```

```
    [47 46 42 43 59 58 62 63 45 44 40 41 57 56 60 61 37 ...
    36 32 33 49 48 52 53 39 38 34 35 51 50 54 55 7 6 2 3 ...
    19 18 22 23 5 4 0 1 17 16 20 21 13 12 8 9 25 24 28 29 ...
    15 14 10 11 27 26 30 31]);
end
%% Processing
switch Mode
    case 1
        y=QPSK.step(u);
    case 2
        y=QAM16.step(u);
    case 3
        y=QAM64.step(u);
    otherwise
        error('Invalid Modulation Mode. Use {1,2, or 3}');
end
```

函数 DemodulatorSoft. m 使用软判决解码进行解调。函数包括三个输入声明：接收到的调制符号流（u），当前子帧的噪声变量估计（NoiseVar）和反应调制模式的参数（Mode）。函数计算 LLR 作为输出。考察这两个函数的不同点，可以看到通过设置解调器系统对象的一组属性，包括叫 DecisionMethod 的属性，我们可以执行软判决解调。

Algorithm

MATLAB function

```
function y=DemodulatorSoft(u, Mode, NoiseVar)
%% Initialization
persistent QPSK QAM16 QAM64
if isempty(QPSK)
    QPSK = comm.PSKDemodulator(...
        'ModulationOrder', 4, ...
        'BitOutput', true, ...
        'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
        'CustomSymbolMapping', [0 2 3 1],...
        'DecisionMethod', 'Approximate log-likelihood ratio', ...
        'VarianceSource', 'Input port');
    QAM16 = comm.RectangularQAMDemodulator(...
        'ModulationOrder', 16, ...
        'BitOutput', true, ...
        'NormalizationMethod', 'Average power', 'SymbolMapping', 'Custom', ...
        'CustomSymbolMapping', [11 10 14 15 9 8 12 13 1 0 4 5 3 2 6 7],...
        'DecisionMethod',  'Approximate log-likelihood ratio', ...
        'VarianceSource', 'Input port');

    QAM64 = comm.RectangularQAMDemodulator(...
    'ModulationOrder', 64, ...
```

```
    'BitOutput', true, ...
    'NormalizationMethod', 'Average power', 'SymbolMapping', 'Custom', ...
    'CustomSymbolMapping', ...
    [47 46 42 43 59 58 62 63 45 44 40 41 57 56 60 61 37 36 32 33 ...
    49 48 52 53 39 38 34 35 51 50 54 55 7 6 2 3 19 18 22 23 5 4 0 1 ...
    17 16 20 21 13 12 8 9 25 24 28 29 15 14 10 11 27 26 30 31],...
    'DecisionMethod',  'Approximate log-likelihood ratio', ...
     'VarianceSource', 'Input port');

end
%% Processing
switch Mode
    case 1
        y=step(QPSK, u, NoiseVar);
    case 2
        y=step(QAM16,u, NoiseVar);
    case 3
        y=step(QAM64, u, NoiseVar);
    otherwise
        error('Invalid Modulation Mode. Use {1,2, or 3}');
end
```

4.1.2 BER 测量

LTE 使用多种调制方法的动机是在给定传输带宽内提供更高数据速率。带宽利用率的单位是（bit/s）/Hz。比较 QPSK，16QAM 和 64QAM 的带宽利用率分别为 QPSK 的 2 倍和 3 倍。不过，高阶调制方案会更易受到信道噪声影响。比较 QPSK，16QAM 或者 64QAM 这样的调制方案，在给定 BER 概率下需要接收端更高的 Eb/N0 值。

下面的 MATLAB 函数为第一个有可实现性的 LTE PHY 发射端模型。这个简单的系统，包括调制器、解调器、AWGN 信道，计算 BER 和 Eb/N0 比的函数。通过运行这个函数，代入不同的 Eb/N0 的值和改变 ModulationMode 参数，我们可以模拟调制阶数和抗信道噪声能力的关系。第一个函数 chap4_ ex01. m，使用硬判决解调器，第二个函数使用软判决解调。这两个函数中使用的组件，也在以后的程序中常用，它们为

1）变量签名与输入 - 输出声明；

2）用户有效负荷（PHY 的输入）（定义参数 FRM 表示一个数据帧中的输入比特数）；

3）停止条件；

4）发射端、信道模型、接收端，和测量部分；

5）在仿真停止时计算 BER。

Algorithm

MATLAB function

```
function [ber, numBits]=chap4_ex01(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2400;     % Size of bit frame
%% Modulation Mode
% ModulationMode=1;                     % QPSK
% ModulationMode=2;                     % QAM16
ModulationMode=3;                       % QAM64
k=2*ModulationMode;                     % Number of bits per modulation symbol
snr = EbNo + 10*log10(k);
%% Processing loop: transmitter, channel model and receiver
numErrs = 0;
numBits = 0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u  = randi([0 1], FRM,1);                          % Randomly generated input bits
   t0 = Modulator(u, ModulationMode);                 % Modulator
   % Channel
   c0 =  AWGNChannel(t0, snr);                        % AWGN channel
   % Receiver
   r0 =  DemodulatorHard(c0, ModulationMode);         % Demodulator, Hard-decision
   y  = r0(1:FRM);                                    % Recover output bits
   % Measurements
   numErrs    = numErrs + sum(y~=u);                  % Update number of bit errors
   numBits    = numBits + FRM;                        % Update number of bits processed
end
%% Clean up & collect results
ber = numErrs/numBits;                                % Compute Bit Error Rate (BER)
```

当我们改变调制方式从 QPSK 到 16QAM 和 64QAM 时，为了达到可靠的传输质量——也就是说，对给定 BER——E/N 值需要相应逐次提高。这说明在较劣化水平的信道中使用低阶调制方案（如 QPSK），为了降低错误概率，需要牺牲数据速率。在条件较好的信道中使用高阶调制方案（如 64QAM），可以提高数据速率。这个结果在图 4.2 中表示，它由改变不同的 E/N 和调制模式参数代入 chap_ ex01. m 得到。图中也比较了各个调制方案的 BER 的理论值和仿真值曲线。

在第 7 章中，我们将会讨论调度器在不同信道条件下随每个调度过程改变调制方案的各种方法。到现在为止，我们只讨论了调制方案，而没有涉及编码和绕码。下面我们会介绍比特级绕码，介绍它的动机，以及它如何用 MATLAB 执行，然后我们会介绍基于 Turbo 编码的纠错，和使用 CRC 处理的检错机制。

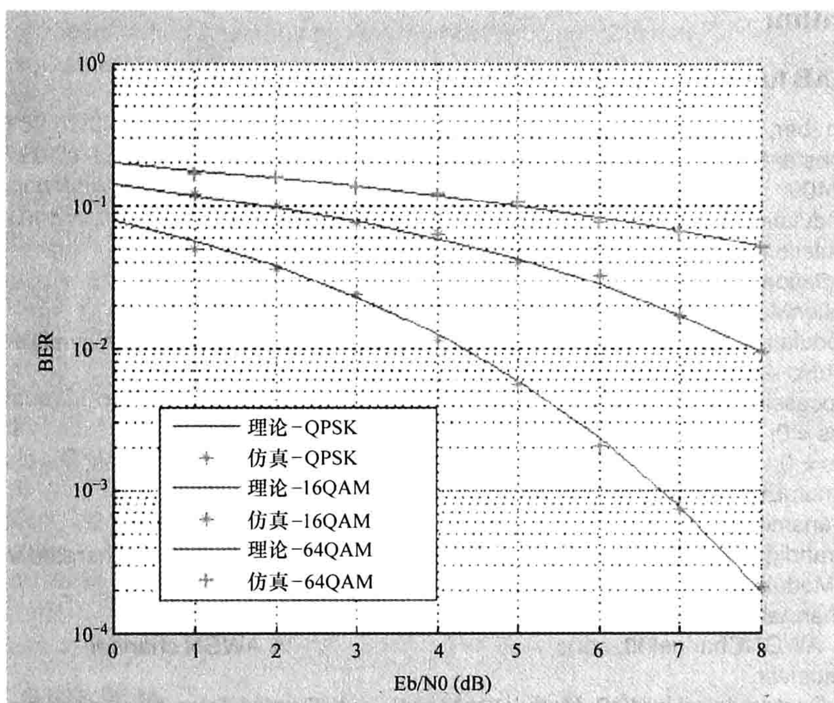


图 4.2 BER 与 Eb/N0 关系：QPSK、16QAM 和 64QAM

4.2 比特级绕码

在 LTE 下行链路中，信道编码操作生成的码字比特由比特级绕码序列绕码。相邻小区使用不同的绕码序列以保证干扰随机，且来自不同小区传输在解码之前被分开。为了达到这些目标，各个小区序列绕码的数据比特唯一，小区的绕码序列由 PHY 小区识别码生成。比特级绕码应用于所有 LTE TrCH 和下行链路控制信道。

绕码由两部分构成：伪随机序列生成和比特相乘。伪随机序列由长度 31 的 Gold 序列生成。输出序列定义为两个序列进行异或操作的输出。两个序列的生成多项式为

$$p_1(x) = x^{31} + x^3 + 1$$
$$p_2(x) = x^{31} + x^3 + x^2 + x + 1 \quad (4.1)$$

第一个序列的初始值定义为长度 31 的单位冲击函数。第二个随机序列的初始值取决于小区识别码、码字数量，和子帧索引。最后，比特相乘是对输入比特和 Gold 多项式比特进行异或运算。绕码其的输出的向量长度与输入码字长度一

致。

在接收端，去绕操作为绕码器的反向过程。它使用相同的伪随机序列生成器。不过，绕码与去扰有所不同。去扰可以通过两种途径执行。假如，先于去扰操作进行硬判决译码，绕码器输入为比特。在这种情况下，输入比特和 Gold 序列的异或运算将会直接生成去扰序输出。另一种途径为，假如在去扰之前进行软判决译码，输入信号不为比特而为 LLR。这种情况下，去扰为输入对数似然值与 Gold 序列比特变换系数值的乘积。一个 0 值 Gold 序列比特映射到 1，而 1 值比特映射到 -1。

4.2.1 MATLAB 实例

下面两个 MATLAB 函数展示如何使用通信系统工具箱组件执行 LTE 绕码和去绕操作。绕码器函数包括两个输入声明：输入比特流（u）和反映当前帧子帧索引的参数（nS）。作为输出，函数计算得到输入比特流的绕码序列。

在绕码器函数中，首先生成一个 Gold 序列产生器系统对象。随后按照 LTE 的准确定义确定 Gold 序列对象的各个属性。两个多项式都按照标准定义设定为 MATLAB 多项式形式。这两个多项式在每个子帧开始时调用 c_ init 变量进行初始化。变量值取决于如小区识别码、码字数和子帧引导符等参数。最后，每个输入采样和 Gold 序列生成器采样相乘得到绕码输出。当绕码输入信号生成信道编码输出比特时，进行异或相乘运算。

Algorithm

MATLAB function

```
function y = Scrambler(u, nS)
% Downlink scrambling
persistent hSeqGen hInt2Bit
if isempty(hSeqGen)
  maxG=43200;
  hSeqGen = comm.GoldSequence('FirstPolynomial',[1 zeros(1, 27) 1 0 0 1],...
                'FirstInitialConditions', [zeros(1, 30) 1], ...
                'SecondPolynomial', [1 zeros(1, 27) 1 1 1 1],...
                'SecondInitialConditionsSource', 'Input port',...
                'Shift', 1600,...
                'VariableSizeOutput', true,...
                'MaximumOutputSize', [maxG 1]);
  hInt2Bit = comm.IntegerToBit('BitsPerInteger', 31);
end
% Parameters to compute initial condition
RNTI = 1;
NcellID = 0;
q =0;
```

```
% Initial conditions
c_init = RNTI*(2^14) + q*(2^13) + floor(nS/2)*(2^9) + NcellID;

% Convert initial condition to binary vector
iniStates = step(hInt2Bit, c_init);

% Generate the scrambling sequence
nSamp = size(u, 1);
seq = step(hSeqGen, iniStates, nSamp);
seq2=zeros(size(u));
seq2(:)=seq(1:numel(u),1);

% Scramble input with the scrambling sequence
y = xor(u, seq2);
```

在去绕函数中，我们使用相同的 Gold 序列生成器反转绕码操作。去绕器初始化与绕码器同步。因为接收器的操作，包括去绕，并没有在标准中定义，我们可以设计两个不同的去扰器。它们的不同之处在于解调生成去绕器输入使用的是软判决还是硬判决方式。

DescramblerSoft 函数生成 LLR 输出，将 Gold 序列比特转换成 -1 和 1 的二进制值，故去绕器的输出也为二进制序列值。

Algorithm

MATLAB function

```
function y = DescramblerSoft(u, nS)
%   Downlink descrambling
persistent hSeqGen hInt2Bit;
if isempty(hSeqGen)
   maxG=43200;
   hSeqGen = comm.GoldSequence('FirstPolynomial',[1 zeros(1, 27) 1 0 0 1],...
                   'FirstInitialConditions', [zeros(1, 30) 1], ...
                   'SecondPolynomial', [1 zeros(1, 27) 1 1 1 1],...
                   'SecondInitialConditionsSource', 'Input port',...
                   'Shift', 1600,...
                   'VariableSizeOutput', true,...
                   'MaximumOutputSize', [maxG 1]);
   hInt2Bit = comm.IntegerToBit('BitsPerInteger', 31);
end
% Parameters to compute initial condition
RNTI = 1;
NcellID = 0;
q=0;
% Initial conditions
c_init = RNTI*(2^14) + q*(2^13) + floor(nS/2)*(2^9) + NcellID;
% Convert to binary vector
```

```
iniStates = step(hInt2Bit, c_init);
% Generate scrambling sequence
nSamp = size(u, 1);
seq = step(hSeqGen, iniStates, nSamp);
seq2=zeros(size(u));
seq2(:)=seq(1:numel(u),1);
% If descrambler inputs are log-likelihood ratios (LLRs) then
% Convert sequence to a bipolar format
seq2 = 1-2.*seq2;
% Descramble
y = u.*seq2;
```

DescramblerHard 函数以硬判决译码进行解调，生成去绕输入。函数对解调输出的比特流和 Gold 序列值进行异或运算（xor）。

Algorithm

MATLAB function

```
function y = DescramblerHard(u, nS)
%   Downlink descrambling
persistent hSeqGen hInt2Bit;
if isempty(hSeqGen)
   maxG=43200;
   hSeqGen = comm.GoldSequence('FirstPolynomial',[1 zeros(1, 27) 1 0 0 1],...
                  'FirstInitialConditions', [zeros(1, 30) 1], ...
                  'SecondPolynomial', [1 zeros(1, 27) 1 1 1 1],...
                  'SecondInitialConditionsSource', 'Input port',...
                  'Shift', 1600,...
                  'VariableSizeOutput', true,...
                  'MaximumOutputSize', [maxG 1]);
   hInt2Bit = comm.IntegerToBit('BitsPerInteger', 31);
end
% Parameters to compute initial condition
RNTI = 1;
NcellID = 0;
q=0;
% Initial conditions
c_init = RNTI*(2^14) + q*(2^13) + floor(nS/2)*(2^9) + NcellID;
% Convert to binary vector
iniStates = step(hInt2Bit, c_init);
% Generate scrambling sequence
nSamp = size(u, 1);
seq = step(hSeqGen, iniStates, nSamp);
% Descramble
y = xor(u(:,1), seq(:,1));
```

4.2.2 BER 测量

下面的部分为用 MATLAB 建模实际 LTE PHY 发射端的第二个函数。在这个例子，chap_ ex02. m 中，我们在调制之前添加绕码操作，在解调之后添加去绕。我们使用软判决解调和其相应的去绕函数，进行软判决输出运算。为了对比输入输出比特，我们转换软判决值为比特值。

Algorithm

MATLAB function

```
function [ber, numBits]=chap4_ex02(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2400;     % Size of bit frame
%% Modulation Mode
% ModulationMode=1;                  % QPSK
% ModulationMode=2;                  % QAM16
ModulationMode=2;                    % QAM64
k=2*ModulationMode;                  % Number of bits per modulation symbol
snr = EbNo + 10*log10(k);
noiseVar = 10.^(0.1.*(-snr));                  % Compute noise variance
%% Processing loop: transmitter, channel model and receiver
numErrs = 0;
numBits = 0;
nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u  = randi([0 1], FRM,1);                              % Randomly generated input bits
   t0 = Scrambler(u, nS);                                    % Scrambler
   t1 = Modulator(t0, ModulationMode);                     % Modulator
   % Channel
   c0 =  AWGNChannel(t1, snr);                               % AWGN channel
   % Receiver
   r0 =  DemodulatorSoft(c0, ModulationMode, noiseVar);  % Demodulator
   r1 =  DescramblerSoft(r0, nS);                            % Descrambler
   y  = 0.5*(1-sign(r1));                                   % Recover output bits
   % Measurements
   numErrs     = numErrs + sum(y~=u);                        % Update number of bit errors
   numBits     = numBits + FRM;                          % Update number of bits processed
   % Manage slot number with each subframe processed
   nS = nS + 2;
   nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;                                % Compute Bit Error Rate (BER)
```

因为绕码操作并不会影响信道噪声灵敏度，前面运行 chap_ ex01. m 的结果

（见图 4.2）与 chap4_ ex02_ m 的差别仅在不同的 E/N 值和调制模式参数上。

表 4.3 传输信道各组件的信道编码方案

传输信道（TrCH）	编码方案	码率
DL - SCH	Turbo 编码	1/3
UL - SCH		
PCH		
MCH		
BCH	尾比特	1/3
	卷积编码	

4.3 信道编码

到目前为止，我们讨论了物理信道调制和绕码操作。现在我们将会转向 TrCH——研究信道编码——附带调制和绕码。我们会介绍基于 Turbo 编码的纠错编码和各种 TrCH 的检错方案。大部分物理信道使用 Turbo 编码，除了广播信道（BCH）——它使用卷积码。

Turbo 编码是 LTE 定义的基本信道编码。Turbo 编码在很多以往的协议中使用过，它一直做为卷积码之外的备选方案。但在 LTE 中，它成为了信道编码机制的核心组件。根据我们的教学进程，我们会分五步，逐步建立 LTE 标准的 TrCH。首先，我们会构建 1/3 码率的 Turbo 编码算法。然后我们会在 Turbo 译码器中添加早期终止机制。它会限定 Turbo 译码器的可计算复杂度。然后我们会介绍速率匹配，它可用 1/3 码率 Turbo 编码器输出进行任意码率的编码。我们会引入与子块分割和码字重组有关的函数。最后，我们将所有组件集成到 TrCH 处理链。在本书中，我们不介绍有关 HARQ 的 MATLAB 函数。HARQ 非常重要，它大幅度减少了回传的数量，提高传输块检错的性能。但如我们在开始时对本书涉及范围的说明，我们关注稳态用户层处理。因此在本书中不讨论 HARQ。

4.4 Turbo 编码

Turbo 编码器属于并行卷积信道编码算法[2]。正如其名，Turbo 编码由两个卷积编码器并联，并由交织器分割。LTE 选择 Turbo 编码基于很多因素。首先，是 Turbo 编码器近 Shannon 极限的性能。在给定足够大 Turbo 译码迭代次数下，Turbo 编码的 BER 性能远远超过传统的卷积码。不仅如此，它拥有适应性，可用于先进的速率匹配机制中，这将在下文中详细讨论。

4.4.1 Turbo 编码器

如图 4.3 所示，LTE 使用 1/3 码率的 Turbo 编码器作为信道编码方案的基础。LTE Turbo 编码器由两个 8 态卷积编码器并联，并由交织器一分为二。Turbo 编码器的输出为三个比特流。第一个的比特通常是系统比特。第二个的比特——也就是两个卷积编码器的输出——通常分别为奇偶校验 1 和奇偶校验 2 比特流。每个卷积编码器由尾比特分隔。这意味着对一个长度 Kbit 的比特流，Turbo 编码器输出三个 K + 4bit 长度的比特流。这会造成 Turbo 编码的码率会略小于 1/3。尾比特会在每个流的末尾相乘，故系统比特和奇偶校验比特流的长度都为 K + 4bit。

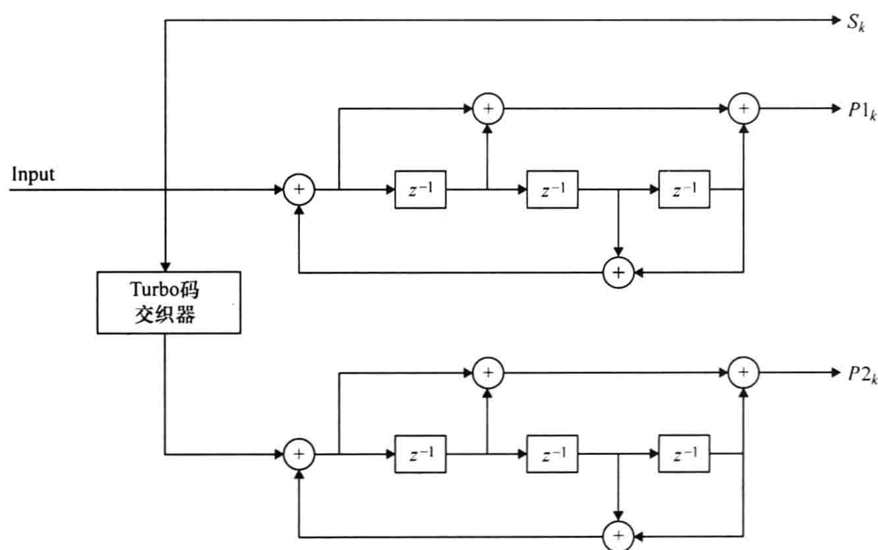


图 4.3　Turbo 编码器的结构图

为了完整定义 Turbo 编码器，我们需要定义卷积编码器的网格结构和 Turbo 编码内交织器。LTE 交织器由正交多项式序列（QPP）构成。交织器转换输入比特。输出比特指数 $p(i)$ 和输入指数 i 之间的关系可以由下面的正交多项式表示：

$$p(i) = (f_1 \cdot i + f_2 \cdot i^2)\bmod(k) \tag{4.2}$$

式中　K 为输入块的长度，f_1 和 f_2 为与 K 值有关的常数。LTE 支持 188 个不同的 K 值长度。最小的长度为 40，最大的 6144。这些块长度和相应的 f_1 和 f_2 常数请参考文献[3]。

LTE Turbo 编码器是一种使用 QPP 交织的无竞争编码器，它通过在交织过程中流水线式访问内存提高编码器性能。卷积编码器的网格结构由下面两个多项式

表示：

$$G_0(z) = 1 + z^{-2} + z^{-3}$$
$$G_1(z) = 1 + z^{-1} + z^{-3} \tag{4.3}$$

它描述了一个 1/3Turbo 编码器有 3 个状态，且其网格结构可用两个前馈和反馈多项式描述，它各拥有 13 和 15 倍频程。

4.4.2 Turbo 译码器

在接收器端，Turbo 译码器进行 Turbo 编码器的反向操作。一个 Turbo 译码器使用两个后验概率（APP）译码器和两个交织器构成反馈环。鉴于存在相同的交织器，APP 译码器拥有和 Turbo 编码器相同的网格结构。不同的地方在于 Turbo 译码器进行迭代运算。Turbo 译码器的可计算复杂度与迭代次数直接相关。

在接收器端，Turbo 译码器进行 Turbo 编码器的反向操作。通过处理来自解调器和去绕器输出的输入信号，Turbo 译码器复原 TrCH 发送比特的最优估计。注意 Turbo 解码器输入需要由 LLR 表示。如我们前文所述，LLR 由使用软判决解调的解调器生成。

4.4.3 MATLAB 实例

下面两个 MATLAB 函数说明了 LTE Turbo 编码器和译码器的定义及其执行。它们使用通信系统工具箱的系统对象。在 TurboEncoder 函数中，我们使用 comm. TurboEncoder 系统对象，并设置网格结构和交织器属性以符合 LTE 标准的定义。通过调用系统对象单步方法，我们将输入比特转化为 Turbo 编码比特输出。

Algorithm

MATLAB function

```
function y=TurboEncoder(u, intrlvrIndices)
%#codegen
persistent Turbo
if isempty(Turbo)
  Turbo = comm.TurboEncoder('TrellisStructure', poly2trellis(4, [13 15], 13), ...
      'InterleaverIndicesSource','Input port');
end
y=step(Turbo, u, intrlvrIndices);
```

同样，TurboDecoder 函数对第一个输入信号（u），即解调器和去绕器的 LLR 输出，进行处理。Turbo 译码器得到发送比特的最优估计。函数同时调用交织指数（intrlvrIndices）和译码器最大迭代次数（maxlter）两个参数作为输入。

Algorithm

MATLAB function

```
function y=TurboDecoder(u, intrlvrIndices, maxIter)
%#codegen
persistent Turbo
if isempty(Turbo)
   Turbo = comm.TurboDecoder('TrellisStructure', poly2trellis(4, [13 15], 13),...
       'InterleaverIndicesSource','Input port', ...
      'NumIterations', maxIter);
end
y=step(Turbo, u,  intrlvrIndices);
```

我们使用通信系统工具箱中的 ploy2trellis 函数设置网格结构。这个函数将译码器连接多项式转化成网格结构。因为 LTE 网格结构同时包括前馈和反馈多项式，我们首先建立一个二进制数描述这两个多项式，然后将这两个二进制数转换为八进制。通过考察图 4.3 中 Turbo 编码器的区块图，我们可以看到编码器约束长度为 4，生成多项式矩阵[13,15]，反馈多项式迭代次数 13。因此，为了设置网格结构的阶数，我们设置 poly2trellis 函数为 poly2trellis（4，［13，15］，13］）。

我们使用 lteIntrlvrIndices 函数构建使用 QPP 的 LTE 交织器。这个函数可查找 LTE 交织器表中支持的 188 个输入长度，并查找相应的 f1 和 f2 常数，然后计算得到符合标准定义的序列向量。

Algorithm

MATLAB function

```
function indices = IntrlvrIndices(blkLen)
%#codegen
[f1, f2] = getf1f2(blkLen);
Idx      = (0:blkLen-1).';
indices  =  mod(f1*Idx + f2*Idx.^2, blkLen) + 1;
```

系统对象 comm. TurboEncoder 和 comm. TurboDecoder 的是 MATLAB 可直接执行的算法。因此，使用 MATLAB 命令行编辑器，我们可以在这两个系统对象执行时随时了解 MATLAB 代码。创建和管理 MATLAB 系统对象超出本书范围。关于这一话题的更多信息，请参考 MATLAB 文档[4]。为了展示 MATLAB 如何按我们所期望的运行，我们可以检查系统对象 stepimpl 函数。

comm. TurboEncoderstepimpl 函数包括两个卷积编码运算，一个对输入信号，一个对交织信号。它随后在网格终点计算额外采样，然后将它们附加在系统比特

和校验比特流的尾部。comm. TurboDecoderstepimpl 重复这一系列操作，包括两个APP 译码器和交织器，*N* 次迭代。*N* 即为 Turbo 译码器的最大迭代次数。在每次迭代的结束，Turbo 译码器更新最优估计的结果。

4.4.4 BER 测量

Turbo 解码运算的性能取决于迭代次数。这意味着对一个给定的 Turbo 编码器（如，LTE 定义的一个编码器），越高的迭代次数对应越好的 BER 性能。函数 chap4_ ex03_ nIter 函数通过计算不同迭代次数的 BER 值表现了这一点。

Algorithm

MATLAB function

```
function [ber, numBits]=chap4_ex03_nIter(EbNo, maxNumErrs, maxNumBits, nIter)
%% Constants
FRM=2432;                                    % Size of bit frame
Indices = lteIntrlvrIndices(FRM);
M=4;k=log2(M);
R= FRM/(3* FRM + 4*3);
snr = EbNo + 10*log10(k) + 10*log10(R);
noiseVar = 10.^(-snr/10);
ModulationMode=1;                            % QPSK
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
  u  =  randi([0 1], FRM,1);                           % Randomly generated input bits
   t0 = TurboEncoder(u, Indices);                          % Turbo Encoder
   t1 = Scrambler(t0, nS);                                 % Scrambler
   t2 = Modulator(t1, ModulationMode);                     % Modulator
   % Channel
   c0 = AWGNChannel(t2, snr);                              % AWGN channel
   % Receiver
   r0 = DemodulatorSoft(c0, ModulationMode, noiseVar);     % Demodulator
   r1 = DescramblerSoft(r0, nS);                           % Descrambler
   y  = TurboDecoder(-r1, Indices,   nIter);               % Turbo Decoder
   % Measurements
  numErrs    = numErrs + sum(y~=u);                   % Update number of bit errors
  numBits    = numBits + FRM;                         % Update number of bits processed
   % Manage slot number with each subframe processed
   nS = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;                          % Compute Bit Error Rate (BER)
```

为了比较 Turbo 编码器和传统卷积编码器的性能，我们同时执行了使用 1/3

码率的卷积编码器函数 chap4_ex03_viterbi. m，它使用 Viterbi 译码和软判决解调。

Algorithm

MATLAB function

```
function [ber, numBits]=chap4_ex03_viterbi(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2432;                          % Size of bit frame
M=4;k=log2(M);
R= FRM/(3* (FRM+6));
snr = EbNo + 10*log10(k) + 10*log10(R);
noiseVar = 10.^(-snr/10);
ModulationMode=1;                  % QPSK
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u = randi([0 1], FRM,1);                          % Randomly generated input bits
   t0 = ConvolutionalEncoder(u);                          % Convolutional Encoder
   t1 = Scrambler(t0, nS);                                 % Scrambler
   t2 = Modulator(t1, ModulationMode);                    % Modulator
   % Channel
   c0 = AWGNChannel(t2, snr);                              % AWGN channel
   % Receiver
   r0 = DemodulatorSoft(c0, ModulationMode, noiseVar);     % Demodulator
   r1 = DescramblerSoft(r0, nS);                           % Descrambler
   r2  = ViterbiDecoder(r1);                               % Viterbi Deocder
   y=r2(1:FRM);
   % Measurements
   numErrs    = numErrs + sum(y~=u);                  % Update number of bit errors
   numBits    = numBits + FRM;                     % Update number of bits processed
   % Manage slot number with each subframe processed
   nS = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;                        % Compute Bit Error Rate (BER)
```

图 4.4 对比了两种译码器的 1 次、3 次，和 5 次迭代的性能。随着我们从 1 到 5 增加迭代次数，我们可以发现 BER 曲线反应了 Turbo 译码器近乎完美的质量。曲线在一个特定的 E/N 值之后成量级下降。比如，在最大迭代数为 5 的情况向，LTE Turbo 译码器附带 QPSK 和软判决解调器可以达到 2e－4 的 BER 值和 1.25dB 的 SNR。

Turbo 编码的这一性能可以解释为什么 LTE 标准选择 Turbo 编码作为用户数据的主要信道编码机制。

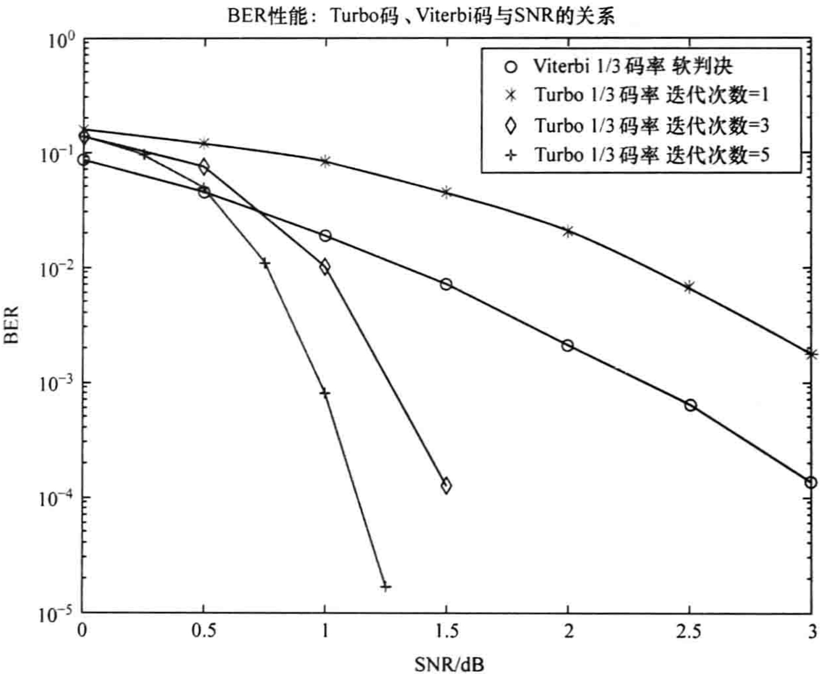


图 4.4　Turbo 码随着迭代次数增加而性能改变

通过执行下面的这个测试平台（chap4_ex03_nIter），我们可以通过测量接收端运算时间得到它与迭代次数的关系。运算时间是估计 Turbo 编码和译码运算可计算复杂度的一个指标。

Algorithm

MATLAB script

```
%% Computation time of turbo coder
%% as a function of number of iterations
EbNo=1;
maxNumErrs=1e6;
maxNumBits=1e6;
for nIter=1:6
    clear functions
    tic;
    ber=chap4_ex03_nIter(EbNo, maxNumErrs, maxNumBits , nIter);
    toc;
end
```

表 4.4 总结了运算时间的结果。如我们预期，复杂度及其译码操作花费的时间随着迭代次数成指数增加。

表 4.4 收发端运算时间与迭代次数的关系

Turbo 编码最大迭代次数	运行时间/s
1	5.83
2	8.54
3	11.27
4	13.66
5	16.41
6	18.96

Algorithm

MATLAB script

```
%% Profiling the turbo coder system model
EbNo=1;
maxNumErrs=1e6;
maxNumBits=1e6;
profile on
ber=chap4_ex03_nIter(EbNo, maxNumErrs, maxNumBits , 1);
profile viewer
```

系统模型每一行的执行时间总结在图 4.5 的报告中。结果显示固定迭代次数的 Turbo 译码花费了整个系统仿真时间的 86%。因此 Turbo 译码可以认为是系统的一个瓶颈。为了克服这一问题，LTE 标准在 LTE 编码器中提供了一种机制，可以早期终止 Turbo 译码从而避免一系列性能上的影响。下节我们将会讨论这一早期终止机制。

Profile Summary

Generated 08-Jul-2013 17:07:57 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
chap4_ex03_nIter	1	18.580 s	0.030 s	
TurboDecoder	412	15.990 s	0.030 s	
TurboDecoder>TurboDecoder.stepImpl	412	15.960 s	15.960 s	
GoldSequence>GoldSequence.stepImpl	824	1.010 s	0.040 s	
GoldSequenceXor>GoldSequenceXor.stepImpl	824	0.970 s	0.950 s	
AWGNChannel	412	0.840 s	0.040 s	
DescramblerSoft	412	0.600 s	0.030 s	
Scrambler	412	0.520 s	0.080 s	

图 4.5 系统模型概况总结，反映 Turbo 译码器是其瓶颈

4.5 早期终止机制

Turbo 译码器的迭代次数是其译码器主要特性之一。为了 Turbo 译码器有效执行，我们很明显面对一个折中。一方面，Turbo 译码器的精度和性能与迭代次数直接相关。更多次迭代得到更好的精度。另一方面，Turbo 译码器可计算复杂度也随着迭代次数增加成指数增长。

LTE 标准提供了一个有效的途径，通过引入早期终止机制解决这一折中。该机制集成在 Turbo 编码器中。通过对 Turbo 编码器输入添加 CRC 校验，我们可以在 Turbo 译码器迭代结束时检出是否存在错误比特。我们可以在 CRC 校验无错误比特时选择提前终止译码，而不需要完成全部译码迭代。这一简单的方案大幅降低了 Turbo 译码器的计算复杂度，且不会带来性能的损失。

4.5.1 MATLAB 实例

下面这个 MATLAB 函数（TurboDecoder_crc）为执行 LTE turbo 译码器时在输入帧结尾检查 CRC 比特以确定是否在所有迭代完成之前终止译码操作。如我们所看到的，在该函数中我们使用 LTETurboDecoder 系统对象而不是 comm. TurboDecoder 系统对象。

Algorithm

MATLAB function

```
function [y, flag, iters]=TurboDecoder_crc(u, intrlvrIndices)
%#codegen
MAXITER=6;
persistent Turbo
if isempty(Turbo)
   Turbo = commLTETurboDecoder('InterleaverIndicesSource', 'Input port', ...
      'MaximumIterations', MAXITER);
end
[y, flag, iters] = step(Turbo, u,  intrlvrIndices);
```

在 LTETurboDecoder 中，执行了与标准 turbo 译码器相似的操作。不过，在每次译码迭代时，对最后 24 个输出采样对应的 CRC 比特进行校验以确定是否有错误。假如没有错误被查出，我们跳出循环并终止 Turbo 译码运算。在此情况，即使并没有执行最大次数的迭代，也可以执行早期终止。假如在 CRC 校验位检出错误，我们会继续译码迭代运算直到最大迭代数完成并没有错误检出。

下面的 MATLAB 函数（CbCRCGenerator）在 Turbo 编码输入的传输块结尾附加了 24bit 的 CRC。

Algorithm

MATLAB function

```
function y = CbCRCGenerator(u)
%#codegen
persistent hTBCRCGen
if isempty(hTBCRCGen)
   hTBCRCGen = comm.CRCGenerator('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
end
% Transport block CRC generation
y = step(hTBCRCGen, u);
```

下面这个 MATLAB 函数（CbCRCDetector）在译码之后在传输块尾部解出 24bit 的 CRC。

Algorithm

MATLAB function

```
function y = CbCRCDetector(u)
%#codegen
persistent hTBCRC
if isempty(hTBCRC)
   hTBCRC = comm.CRCDetector('Polynomial',  [1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
end
% Transport block CRC generation
y = step(hTBCRC, u);
```

4.5.2　BER 测量

为了考察早期终止算法的有效性，我们现在对比 Turbo 译码器附加 CRC 终止机制和不附加终止机制的结果。下面这个函数（chap4_ex04）包含了 CRC 生成、Turbo 编码、绕码，和调制及其反过程，但不包括早期终止机制。

Algorithm

MATLAB function

```
function [ber, numBits]=chap4_ex04(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2432-24;                                  % Size of bit frame
Kplus=FRM+24;
Indices = lteIntrlvrIndices(Kplus);
ModulationMode=1;
```

```
k=2*ModulationMode;
maxIter=6;
CodingRate=Kplus/(3*Kplus+12);
snr = EbNo + 10*log10(k) + 10*log10(CodingRate);
noiseVar = 10.^(-snr/10);
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
  % Transmitter
  u = randi([0 1], FRM,1);                          % Randomly generated input bits
  data= CbCRCGenerator(u);                          % Code block CRC generator
  t0 = TurboEncoder(data, Indices);                 % Turbo Encoder
  t1 = Scrambler(t0, nS);                           % Scrambler
  t2 = Modulator(t1, ModulationMode);               % Modulator
  % Channel
  c0 = AWGNChannel(t2, snr);                        % AWGN channel
  % Receiver
  r0 = DemodulatorSoft(c0, ModulationMode, noiseVar);  % Demodulator
  r1 = DescramblerSoft(r0, nS);                     % Descrambler
  r2  = TurboDecoder(-r1, Indices,  maxIter);       % Turbo Decoder
  y  = CbCRCDetector(r2);                           % Code block CRC detector
  % Measurements
  numErrs   = numErrs + sum(y~=u);                  % Update number of bit errors
  numBits   = numBits + FRM;                        % Update number of bits processed
  % Manage slot number with each subframe processed
  nS = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;                              % Compute Bit Error Rate (BER)
```

下面这个函数（chap_ex04_crc）包括相同的接收端并包括早期终止机制。在此包含早期终止的算法中，我们记录每个帧的实际迭代次数并统计直方图。

Algorithm

MATLAB function

```
function [ber, numBits, itersHist]=chap6_ex04_crc(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2432-24;                                        % Size of bit frame
Kplus=FRM+24;
Indices = lteIntrlvrIndices(Kplus);
ModulationMode=1;
k=2*ModulationMode;
maxIter=6;
CodingRate=Kplus/(3*Kplus+12);
snr = EbNo + 10*log10(k) + 10*log10(CodingRate);
```

```
noiseVar = 10.^(-snr/10);
Hist=dsp.Histogram('LowerLimit', 1, 'UpperLimit', maxIter, 'NumBins', maxIter,
'RunningHistogram', true);
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u = randi([0 1], FRM,1);                          % Randomly generated input bits
   data= CbCRCGenerator(u);                           % Transport block CRC code
   t0 = TurboEncoder(data, Indices);                   % Turbo Encoder
   t1 = Scrambler(t0, nS);                             % Scrambler
   t2 = Modulator(t1, ModulationMode);                % Modulator
   % Channel
   c0 = AWGNChannel(t2, snr);                          % AWGN channel
   % Receiver
   r0 = DemodulatorSoft(c0, ModulationMode, noiseVar);    % Demodulator
   r1 = DescramblerSoft(r0, nS);                       % Descrambler
   [y, ~, iters] = TurboDecoder_crc(-r1, Indices);     % Turbo Decoder
   % Measurements
   numErrs    = numErrs + sum(y~=u);                  % Update number of bit errors
   numBits    = numBits + FRM;                      % Update number of bits processed
   itersHist  =  step(Hist, iters);                   % Update histogram
of iteration numbers
   % Manage slot number with each subframe processed
   nS = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;                             % Compute Bit Error Rate (BER)
```

图 4.6 的 BER 结果显示引入早期终止机制（红线）与否（蓝线）对一定范围的 SNR 值不影响 BER 性能。

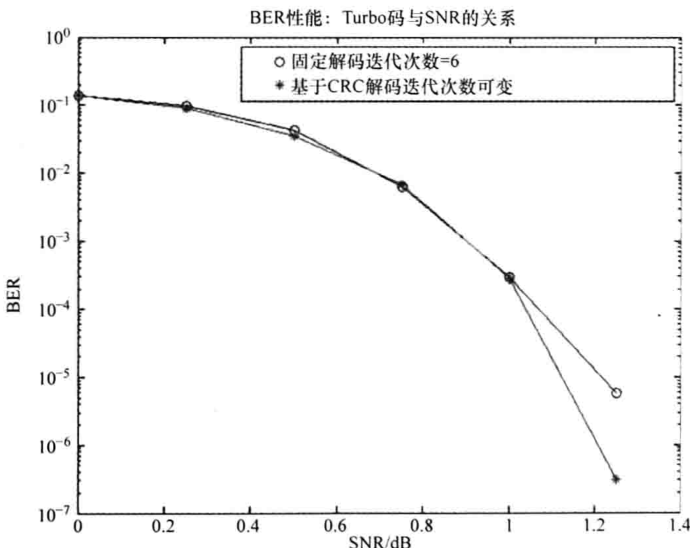


图 4.6　对比 CRC 早期终止机制引入与否的 Turbo 编码 BER 性能

4.5.3 计时测量

在本例中，我们比较接收端使用 CRC 早期终止机制（chap_ ex04_ crc. m）与不使用早期终止机制（chap_ ex04. m）的运行时间。该测试调用如下 MATLAB 测试平台。

Algorithm

MATLAB script

```
EbNo=1; maxNumErrs=1e7; maxNumBits=1e7;
tic; [a,b]=chap4_ex04(EbNo,maxNumErrs, maxNumBits); toc;
tic; [a,b]=chap4_ex04_crc(EbNo,maxNumErrs, maxNumBits); toc;
```

脚本第一行驱动两个函数在 Eb/N0 值为 1dB 的要求下处理 10 万比特。第二行使用 MATLAB 函数 tic 和 toc 得到不使用早期终止机制的程序运行时间。第三行记录使用早期终止机制的程序运行时间。

MATLAB 命令行显示结果如图 4.7 所示。在相同的输入帧数情况下，使用早期终止机制的程序运行时间（131. 27s）明显小于不使用早期终止的程序运行时间（178. 77s）。

```
>> EbNo=1; maxNumErrs=1e7; maxNumBits=1e7;
tic; [a,b]=chap4_ex04(EbNo,maxNumErrs, maxNumBits); toc
tic; [a,b]=chap4_ex04_crc(EbNo,maxNumErrs, maxNumBits); toc
Elapsed time is 178.771266 seconds.
Elapsed time is 131.273779 seconds.
```

图 4.7 早期终止机制明显节省 Turbo 译码时间

4.6 码率匹配

以上我们只考虑了 1/3 码率的 Turbo 码操作。码率匹配作为现代通信标准的一个重要特征，在适应性编码中应用。它可以帮助提高不同信道条件下的数据吞吐量。在低劣化信道中，我们用近似单位码率编码数据，减小发送前向纠错编码的比特数量。另一方面，在高劣化信道，我们可以使用更小的码率并增加检错比特。

在码率匹配的信道编码中，我们以 1/3 码率 Turbo 编码为基础并使用码率匹配通过重复或移除的方式得到任意所需的码率。假如所需码率低于 1/3，我们就对 Turbo 编码器输出比特进行重复。对大于 1/3 的码率，我们减少或移除一些 Turbo 编码器输出比特。移除编码并不是简单二次采样的结果，它基于交织方

法。这个方法维持高码率编码的 hamming 间距。

码率匹配包括：

1）子块交织；

2）奇偶校验比特隔行；

3）比特裁剪；

4）码率比特选择与传输。

码率匹配的第一个操作时子块交织，它使用一个单矩形交织器。通过在码率匹配中使用循环缓冲器，裁剪和重复操作增加或减少码率得到需要的水平可在循环缓冲器中进行比特选择操作简单实现。最后，通过连接码组，编码比特完成并发送至 PDSCH 处理。

4.6.1 MATLAB 实例

为了继续我们从简单到复杂的教学过程，我们在介绍 LTE 标准中所有传输块信道编码细节之前首先研究码率匹配。这个 MATLAB 函数实现 LTE 标准定义的三个码率匹配的特征：子块交织、奇偶校验比特隔行，和包括循环缓冲器比特收集的码率匹配。

MATLAB 函数包括一系列 LTE 码率匹配算法定义的交织、隔行，和比特收集操作。码率匹配的输入时 1/3Turbo 编码器的输出。这样，对于长度为 K 的输入块，码率匹配器的输入为 $3(K+4)$，包括系统和两个奇偶校验共三个比特流。首先，我们分隔每三个流为 32bit 的区块，并对它们进行交织。考虑到每个流可能无法每 32bit 分隔，我们在每个流的开始增加一些哑比特，以使向量可以分隔成整 32bit。子块交织过程对系统和奇校验比特相同，但对偶校验比特不同。

我们随后生成包括附加哑比特的系统比特和隔行奇偶校验比特的输出向量。最后，通过移除哑比特，我们生成循环缓冲器进行码率收缩。码率匹配的最后一步是比特收集，在此步骤中循环缓冲器中的哑比特被移除，最初的几个比特被收集。收集比特与 Turbo 编码器输入长度的比值就是码率匹配之后的新码率。

Algorithm

MATLAB function

```
function y= RateMatcher(in, Kplus, Rate)
% Rate matching per coded block, with and without the bit selection.
D = Kplus+4;
if numel(in)~=3*D, error('Kplus (2nd argument) times 3 must be size of input 1.');end
% Parameters
colTcSb = 32;
rowTcSb = ceil(D/colTcSb);
Kpi = colTcSb * rowTcSb;
Nd = Kpi - D;
```

```
% Bit streams
d0 = in(1:3:end); % systematic
d1 = in(2:3:end); % parity 1st
d2 = in(3:3:end); % parity 2nd
i0=(1:D)';
Index=indexGenerator(i0,colTcSb, rowTcSb, Nd);
Index2=indexGenerator2(i0,colTcSb, rowTcSb, Nd);

% Sub-block interleaving - per stream
v0 = subBlkInterl(d0,Index);
v1 = subBlkInterl(d1,Index);
v2 = subBlkInterl(d2,Index2);
vpre=[v1,v2].';
v12=vpre(:);
%   Concat 0, interleave 1, 2 sub-blk streams
% Bit collection
wk = zeros(numel(in), 1);
wk(1:D) = remove_dummy_bits( v0 );
wk(D+1:end) = remove_dummy_bits( v12);
% Apply rate matching
N=ceil(D/Rate);
y=wk(1:N);
end
function v = indexGenerator(d, colTcSb, rowTcSb, Nd)
% Sub-block interleaving - for d0 and d1 streams only
colPermPat = [0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30,...
          1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31];
% For 1 and 2nd streams only
y = [NaN*ones(Nd, 1); d];      % null (NaN) filling
inpMat = reshape(y, colTcSb, rowTcSb).';
permInpMat = inpMat(:, colPermPat+1);
v = permInpMat(:);
end
function v = indexGenerator2(d, colTcSb, rowTcSb, Nd)
% Sub-block interleaving - for d2 stream only
colPermPat = [0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30,...
          1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31];
pi = zeros(colTcSb*rowTcSb, 1);
for i = 1 : length(pi)
   pi(i) = colPermPat(floor((i-1)/rowTcSb)+1) + colTcSb*(mod(i-1, rowTcSb)) + 1;
end
% For 3rd stream only
y = [NaN*ones(Nd, 1); d];      % null (NaN) filling
inpMat = reshape(y, colTcSb, rowTcSb).';
ytemp = inpMat.';
y = ytemp(:);
v = y(pi);
end
function out = remove_dummy_bits( wk )
%UNTITLED5 Summary of this function goes here
```

```
%out = wk(find(~isnan(wk)));
out=wk(isfinite(wk));
end
function out=subBlkInterl(d0,Index)
out=zeros(size(Index));
IndexG=find(~isnan(Index)==1);
IndexB=find(isnan(Index)==1);
out(IndexG)=d0(Index(IndexG));
Nd=numel(IndexB);
out(IndexB)=nan*ones(Nd,1);
end
```

在 RateDematcher 函数中我们反向码率匹配的所有操作。我们创建了一个包括哑比特、系统以及奇偶校验比特的向量，将输入向量的采样放置其中，并通过隔行和去隔行建立 LLR 采样的合适值作为 1/3Turbo 译码器的输入。

Algorithm

MATLAB function

```
function out = RateDematcher(in, Kplus)
% Undoes the Rate matching per coded block.
%#codegen

% Parameters
colTcSb = 32;
D = Kplus+4;
rowTcSb = ceil(D/colTcSb);
Kpi = colTcSb * rowTcSb;
Nd = Kpi - D;

tmp=zeros(3*D,1);
tmp(1:numel(in))=in;

% no bit selection - assume full buffer passed in
i0=(1:D)';
Index= indexGenerator(i0,colTcSb, rowTcSb, Nd);
Index2= indexGenerator2(i0,colTcSb, rowTcSb, Nd);
Indexpre=[Index,Index2+D].';
Index12=Indexpre(:);

% Bit streams
tmp0=tmp(1:D);
tmp12=tmp(D+1:end);
v0 = subBlkDeInterl(tmp0, Index);
d12=subBlkDeInterl(tmp12, Index12);
v1=d12(1:D);
v2=d12(D+(1:D));
```

```
% Interleave 1, 2, 3 sub-blk streams - for turbo decoding
temp = [v0 v1 v2].';
out = temp(:);
end

function v = indexGenerator(d, colTcSb, rowTcSb, Nd)
% Sub-block interleaving - for d0 and d1 streams only

colPermPat = [0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30,...
           1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31];

% For 1 and 2nd streams only
y = [NaN*ones(Nd, 1); d];       % null (NaN) filling
inpMat = reshape(y, colTcSb, rowTcSb).';
permInpMat = inpMat(:, colPermPat+1);
v = permInpMat(:);

end

function v = indexGenerator2(d, colTcSb, rowTcSb, Nd)
% Sub-block interleaving - for d2 stream only

colPermPat = [0, 16, 8, 24, 4, 20, 12, 28, 2, 18, 10, 26, 6, 22, 14, 30,...
           1, 17, 9, 25, 5, 21, 13, 29, 3, 19, 11, 27, 7, 23, 15, 31];
pi = zeros(colTcSb*rowTcSb, 1);
for i = 1 : length(pi)
   pi(i) = colPermPat(floor((i-1)/rowTcSb)+1) + colTcSb*(mod(i-1, rowTcSb)) + 1;
end

% For 3rd stream only
y = [NaN*ones(Nd, 1); d];       % null (NaN) filling
inpMat = reshape(y, colTcSb, rowTcSb).';
ytemp = inpMat.';
y = ytemp(:);
v = y(pi);

end
function out=subBlkDeInterl(in,Index)
out=zeros(size(in));
IndexG=find(~isnan(Index)==1);
IndexOut=Index(IndexG);
out(IndexOut)=in;
end
```

4.6.2 BER 测量

我们现在将考察 Turbo 编码算法使用不同于 1/3 的其他码率的影响。函数 chap6_ex05_crc 为收发端算法，包括了 CRC 生成、Turbo 编码、绕码和调制及其饭过程，并应用早期终止机制和码率匹配。

Algorithm

MATLAB function

```
function [ber, numBits, itersHist]=chap6_ex05_crc(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2432-24;                              % Size of bit frame
Kplus=FRM+24;
Indices = lteIntrlvrIndices(Kplus);
ModulationMode=1;
k=2*ModulationMode;
CodingRate=1/2;
snr = EbNo + 10*log10(k) + 10*log10(CodingRate);
noiseVar = 10.^(-snr/10);
Hist=dsp.Histogram('LowerLimit', 1, 'UpperLimit', maxIter, 'NumBins', maxIter,
'RunningHistogram', true);
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
  % Transmitter
  u  =  randi([0 1], FRM,1);                          % Randomly generated input bits
  data= CbCRCGenerator(u);                              % Transport block CRC code
  t0 = TurboEncoder(data, Indices);                      % Turbo Encoder
  t1= RateMatcher(t0, Kplus, CodingRate);                % Rate Matcher
  t2 = Scrambler(t1, nS);                                % Scrambler
  t3 = Modulator(t2, ModulationMode);                    % Modulator
  % Channel
  c0 = AWGNChannel(t3, snr);                             % AWGN channel
  % Receiver
  r0 = DemodulatorSoft(c0, ModulationMode, noiseVar);    % Demodulator
  r1 = DescramblerSoft(r0, nS);                          % Descrambler
  r2 = RateDematcher(r1, Kplus);                         % Rate Matcher
  [y, ~, iters]  = TurboDecoder_crc(-r2, Indices);       % Turbo Decoder
  % Measurements
  numErrs    = numErrs + sum(y~=u);                      % Update number of bit errors
  numBits    = numBits + FRM;                            % Update number of bits processed
  itersHist    =  step(Hist, iters);                     % Update histogram
of iteration numbers
  % Manage slot number with each subframe processed
  nS = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;                               % Compute Bit Error Rate (BER)
```

通过在 Turbo 编码器之前添加码率匹配以及在译码器之前添加码率去匹配操作，我们可以仿真出任意不同于 1/3 码率的情况，较低码率在传输中用于较好的信道条件，它只需要较少的纠错过程。

通过更新函数中的变量 CodingRate，我们可以进行码率匹配操作并考察多种

码率情况下 BER 随 SNR 变化的情况。其结果显示在图 4.8 中，如我们所预期的，1/3 码率收发端要比 1/2 码率收发端性能好很多。

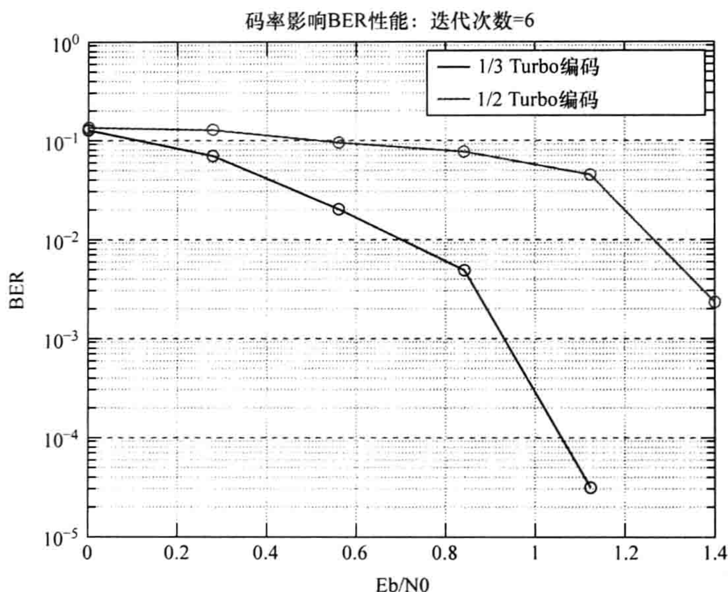


图 4.8 Turbo 编码中进行码率匹配对 BER 性能的影响

4.7 码块分段

在 LTE 中，传输块连接了 MAC 层和 PHY 层。传输块一般包括同时传输的大数据比特。对传输块的第一步操作是对其进行信道编码，它对每个码块独立进行。假如输入帧大于 Turbo 编码器的最大处理长度，我们一般会分割传输块为多个小块，即码块。因为 Turbo 编码器内部的交织器定义最大处理长度 188 的输入块，码块的长度需要和 Turbo 编码器设定的码块长度匹配。随后对每个码块独立进行添加 CRC、Turbo 编码和码率匹配等操作。

4.7.1 MATLAB 实例

在下面这一小段程序里我们寻找最合适的码块长度满足两个属性：

1）最大长度 188；

2）子码块长度为整数。

一个码块内子码块的数量反映在参数 C，而每个子码块的长度反映在参数 Kplus。我们同时需要对码字计算参数 E。信道编码的输出即是码字；码字的长度为子码块参数 C 和每个子码块输出长度 E 的乘积。总码字长度由调度器决定，取决于可用资源数量。有效编码速率即为码字长度与子码块长度的比值。

Algorithm

MATLAB function

```
function [C, Kplus] = CblkSegParams(tbLen)
%#codegen
%% Code block segmentation
blkSize = tbLen + 24;
maxCBlkLen = 6144;
if (blkSize <= maxCBlkLen)
   C = 1;         % number of code blocks
   b = blkSize;   % total bits
else
   L = 24;
   C = ceil(blkSize/(maxCBlkLen-L));
   b = blkSize + C*L;
end

% Values of K from table 5.1.3-3
validK = [40:8:512 528:16:1024 1056:32:2048 2112:64:6144].';
% First segment size
temp = find(validK >= b/C);
Kplus = validK(temp(1), 1);    % minimum K
```

下面这个 MATLAB 函数计算了子码块长度并判断多少子码块需要并行处理以形成信道编码输出。首先，我们根据子码块数量风格总码字比特数，我们由调制比特数确认输出比特数，并最终确定多天线层数。

Algorithm

MATLAB function

```
function E = CbBitSelection(C, G, Nl, Qm)
%#codegen
% Bit selection parameters
% G = total number of output bits
% Nl   Number of layers a TB is mapped to (Rel10)
% Qm    modulation bits
Gprime = G/(Nl*Qm);
gamma = mod(Gprime, C);
E=zeros(C,1);
% Rate matching with bit selection
for cbIdx=1:C
      if ((cbIdx-1) <= (C-1-gamma))
         E(cbIdx) = Nl*Qm*floor(Gprime/C);
      else
         E(cbIdx)   = Nl*Qm*ceil(Gprime/C);
      end
end
```

在接收端，为了正确反向匹配操作，我们需要参数 C 和 Kplus（子码块数和每个子码块长度）。

4.8 LTE 传输信道处理

图 4.9 所示为 TrCH 处理的区块图。五个功能组件构成了传输块处理：

1）传输块添加 CRC；

2）码块分端和码块 CRC 添加；

3）1/3 码率 Turbo 编码；

4）匹配任意所需码率；

5）码块连接。

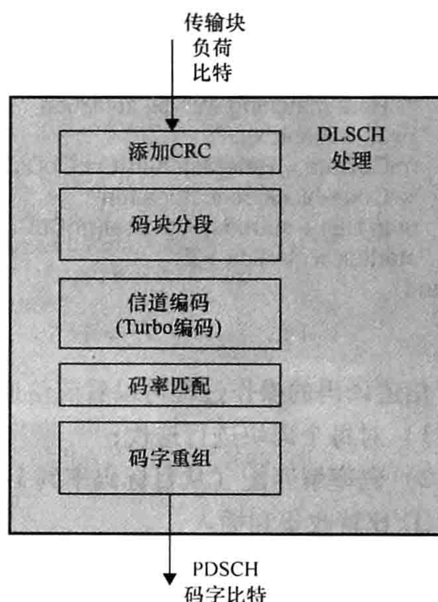


图 4.9 传输信道处理结构

4.8.1 MATLAB 实例

在下面的 MATLAB 函数中，我们需要区别开两种情况：传输块只包含单一码块和传输块包含多个码块，当第一种情况时我们不需要在码字上添加 CRC 因为传输块已经添加了 CRC。

Algorithm

MATLAB function

```
function [out, Kplus, C] = TbChannelCoding(in, prmLTE)
% Transport block channel coding
%#codegen
inLen = size(in, 1);
[C, ~, Kplus] =  CblkSegParams(inLen-24);
intrlvrIndices = lteIntrlvrIndices(Kplus);
G=prmLTE.maxG;
E_CB=CbBitSelection(C, G, prmLTE.NumLayers, prmLTE.Qm);
% Initialize output
out = false(G, 1);
% Channel coding the TB
if (C==1) % single CB, no CB CRC used
   % Turbo encode
   tEncCbData = TurboEncoder( in, intrlvrIndices);
   % Rate matching, with bit selection
   rmCbData =  RateMatcher(tEncCbData, Kplus,  G);
   % unify code paths
```

```
    out = logical(rmCbData);
else % multiple CBs in TB
    startIdx = 0;
    for cbIdx = 1:C
        % Code-block segmentation
        cbData = in((1:(Kplus-24)) + (cbIdx-1)*(Kplus-24));
        % Append checksum to each CB
        crcCbData = CbCRCGenerator( cbData);
        % Turbo encode each CB
        tEncCbData = TurboEncoder(crcCbData, intrlvrIndices);
        % Rate matching with bit selection
        E=E_CB(cbIdx);
        rmCbData = RateMatcher(tEncCbData, Kplus,  E);
        % Code-block concatenation
        out((1:E) + startIdx) = logical(rmCbData);
        startIdx = startIdx + E;
    end
end
```

信道译码的操作过程可以看成信道编码的反向，包括：

1）对每个码块进行迭代；

2）码率解匹配（从目标码率到 1/3）包括：

① 比特收集和插入；

② 奇偶校验解隔行；

③ 子码块解交织；

④ 为 Turbo 译码复原系统比特和奇偶校验比特；

3）码块以 1/3 码率进行带 CRC 早期终止机制的 Turbo 译码。

在这里我们使用特定传输块的 CRC 作为早期终止判决条件和更新 HARQ 状态的机制。下面的 MATLAB 函数总结了 TrCH 译码操作。

Algorithm

MATLAB function

```
function [decTbData, crcCbFlags, iters] = TbChannelDecoding( in, Kplus, C, prmLTE)
% Transport block channel decoding.
%#codegen
intrlvrIndices = lteIntrlvrIndices(Kplus);
% Make fixed size
G=prmLTE.maxG;
E_CB=CbBitSelection(C, G, prmLTE.NumLayers, prmLTE.Qm);
% Channel decoding the TB
if (C==1) % single CB, no CB CRC used
    % Rate dematching, with bit insertion
```

```
    deRMCbData = RateDematcher(-in, Kplus)
    % Turbo decode the single CB
    tDecCbData =TurboDecoder(deRMCbData, intrlvrIndices,  prmLTE.maxIter)
    % Unify code paths
    decTbData  = logical(tDecCbData);
 else % multiple CBs in TB
    decTbData  = false((Kplus-24)*C,1); % Account for CB CRC bits
    startIdx = 0;
    for cbIdx = 1:C
       % Code-block segmentation
       E=E_CB(cbIdx);
       rxCbData = in(dtIdx(1:E) + startIdx);
       startIdx = startIdx + E;
       % Rate dematching, with bit insertion
       %   Flip input polarity to match decoder output bit mapping
       deRMCbData = lteCbRateDematching(-rxCbData, Kplus, C, E);
       % Turbo decode each CB with CRC detection
       %   - uses early decoder termination at the CB level
       [crcDetCbData, crcCbFlags(cbIdx), iters(cbIdx)] = ...
               TurboDecoder_crc(deRMCbData, intrlvrIndices);
       % Check the crcCBFlag per CB. If still in error, abort further TB
       % processing for remaining CBs in the TB, as the HARQ process will
       % request retransmission for the whole TB.
       if (~prmLTE.fullDecode)
          if (crcCbFlags(cbIdx)==1) % error
             break;
          end
       end
       % Code-block concatention
       decTbData((1:(Kplus-24)) + (cbIdx-1)*(Kplus-24)) = logical(crcDetCbData);
    end
end
```

4.8.2 BER 测量

我们现在测量 LTE 下行链路 TrCH 在 AWGN 信道噪声条件下的比特误码率。函数 chap4_ex06 包括了所有 TrCH 处理过程并绕码和调制。

Algorithm

MATLAB function

```
function [ber, numBits]=chap4_ex06(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2432-24;
Kplus=FRM+24;
```

```
Indices = lteIntrlvrIndices(Kplus);
ModulationMode=1;
k=2*ModulationMode;
maxIter=6;
CodingRate=1/2;
snr = EbNo + 10*log10(k) + 10*log10(CodingRate);
noiseVar = 10.^(-snr/10);
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u  =  randi([0 1], FRM,1);                              % Randomly generated input bits
   data= CbCRCGenerator(u);                                   % Transport block CRC code
   [t1, Kplus, C] = TbChannelCoding(data,Indices,maxIter);       % Transport
Channel encoding
   t2 = Scrambler(t1, nS);                                       % Scrambler
   t3 = Modulator(t2, ModulationMode);                          % Modulator
   % Channel
   c0 = AWGNChannel(t3, snr);                                   % AWGN channel
   % Receiver
   r0 = DemodulatorSoft(c0, ModulationMode, noiseVar);          % Demodulator
   r1 = DescramblerSoft(r0, nS);                                 % Descrambler
    [r2,~,~] = TbChannelDecoding(r1, Kplus, C, Indices,maxIter);  % Transport
Channel decoding
   y  =  CbCRCDetector(r2);                                   % Code block CRC detector
   % Measurements
   numErrs    = numErrs + sum(y~=u);                         % Update number of bit errors
  numBits    = numBits + FRM;                             % Update number of bits processed
   % Manage slot number with each subframe processed
   nS = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;                             % Compute Bit Error Rate (BER)
```

通过在一定 SNR 范围内执行该函数，我们可以验证 DLSCH 和 PDSCH 的处理不需要 OFDM 和 MIMO 操作可以得到不错的性能。图 4.10 展示了收发端的 BER 特性。在这个程序中，我们匹配 1/2 码率和 QPSK 调制，以及重复最大迭代数从 1 到 6 进行迭代。如我们预期，早期终止作为一个关键机制使 LTE 定义的 DLSCH 处理更具有可实现性。

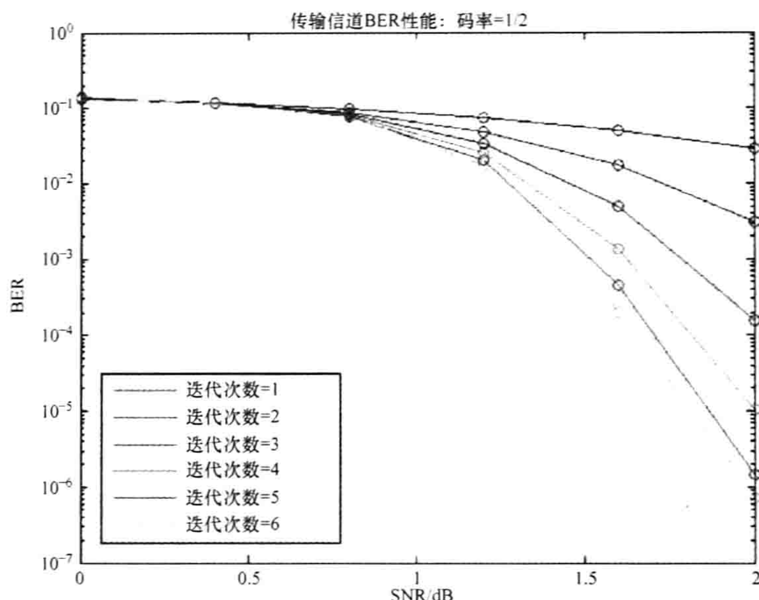


图 4.10 DLSCH 和 QPSK 的 BER 特性与 Eb/N0 和译码操作次数的关系

4.9 本章小结

到此为止，我们基于简单的信道模型（AWGN）研究了 LTE 标准中使用的前馈纠错方案。LTE 标准用 AWGN 环境传输进行静态性能测量。衰落和多径效应在这个传输模型中不做考虑，真实信道的频率响应也不计入其中。大多数真实信道包含了传输信号的各种衰落和相应的劣化。这些衰落造成了码间串扰，它必须使用平衡器进行补偿。

我们考察了 Turbo 信道编码迭代次数与性能的关系。这将成为第 9 章中仿真加速部分讨论的原动力。我们也要注意到 AWGN 信道条件下的研究忽略了真实信道模型和多径衰落效应。这将是第 5 章和第 6 章讨论如何用 OFDM 和频域平衡器单载波频分复用（SC－FDM）解决多径衰落的原动力。

参考文献

[1] 3GPP (2009) Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and Channel Coding. TS 36.212.

[2] Proakis, J. and Salehi, M. (2007) *Digital Communications*, 5th edn, McGraw-Hill Education, New York.

[3] 3GPP (2011) Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation Version 10.0.0. TS 36.211, January 2011.

[4] Online: http://www.mathworks.com/help/comm/ug/define-basic-system-objects-1.html (last accessed September 30, 2013).

5 OFDM

到目前为止，我们已经对 LTE 标准中定义的调制、绕码和编码进行研究并用简单的信道模型进行性能评估。理解正交频分复用这一 LTE 空中接口的基础，对理解和建模更复杂的信道模型至关重要。

在本章中，我们考量加入动态信道响应和衰落条件的现实信道模型。短期衰落效应如多径衰落和由移动造成的多普勒效应会造成信道模型的频率选择性。LTE 中下行链路的 OFDM 和上行链路的单载波频分复用（SC－FDM）多路接入技术，使用高效频域均衡器补偿频率选择性衰落和得到更高的频谱效率。在本章中我们会关注单天线配置，而在下一章中我们会讨论多输入多输出（MIMO）和其 OFDM。

我们详细讲解 OFDM 技术基础并讨论 LTE 标准中的 OFDM 帧结构和实现。我们随后会讨论 OFDM 信号的时－频映射和多种适应信道带宽的资源元素粒度。它们由接收端 OFDM 信号频域均衡器确定。我们会考察迫零（ZF）、最小均方误差（MMSE）、均衡器和重要参考或导频信号。最后，我们考察到现在为止介绍的组件所组成的收发端在 LTE 定义的多种多径衰落和移动条件下的性能。

5.1 信道建模

移动信道可以由发射端和接收端之间多路径传播的有效性描述。出了收发端之间的少之又少的直线路径之外，其他路径由反射、折射、散射，或其他传播途径形成。接收端可以同时收到通过不同路径而状态各不相同的传输信号。这些不同状态的信号携带了变化的信号功率以及时延和相延。因为这些接收到的信号与时间相关，AWGN 模型不适用于大部分无线连接的信道建模。因此，建立更适合描述无线信道的模型对于设计移动通信系统是很重要的。信道传播通常带来接收信号功率衰减。一般情况下，功率衰减分为两类：

1）信号幅度衰减或大尺度衰落；

2）衰落或小尺度衰落。

5.1.1 大尺度和小尺度衰落

路径损耗和阴影衰落效应是大部分大尺度衰落效应的主要因素。这些大尺度特征需在设计和小区拓扑中考虑[1]。小尺度衰落包括多径衰落和移动过程带来的时间色散。这些特征为短时并必须做出适应性处理。PHY 层设计应该包括有

效应对这些信道衰减的技术[1]。

5.1.2 多径衰落效应

多径衰落可由功率延时描述，它包括两个部分：有关延时的向量和有关平均功率参数的向量。其他有用的可测配置既包括有关延时的一阶矩量平均附加延时或二阶矩量均方根（RMS）延时。多径衰落可以是平坦性或频率选择性的。假如带宽大于延时扩散的倒数，信道频率响应就会导致多径衰落。

在小区通信中，移动终端从基站延直线路径接收信号。有些信号也会被建筑物或其他反射源反射，从而在到达移动终端时出现时延和功率损失。因移动接收器线性合并这些信号，总信号实际为信号和信道冲击响应的卷积。在频域，信道频率响应在不同频率有不同的响应形态；因此出现频率选择性衰落（见图 5.1）。

在时间色散信道带有多径传播特性的情况下，子载波不仅仅会出现码内串扰，并会出现载波间串扰。这是因为载波间正交性会因为不同路径下分割码间边界的解码器间隔重叠，而造成部分缺失。因为调制符号在相邻的两个符号间隔上并不相同，用于计算快速傅里叶变换（FFT）的积分区间，并不与一定路径复指数的整数周期一致。

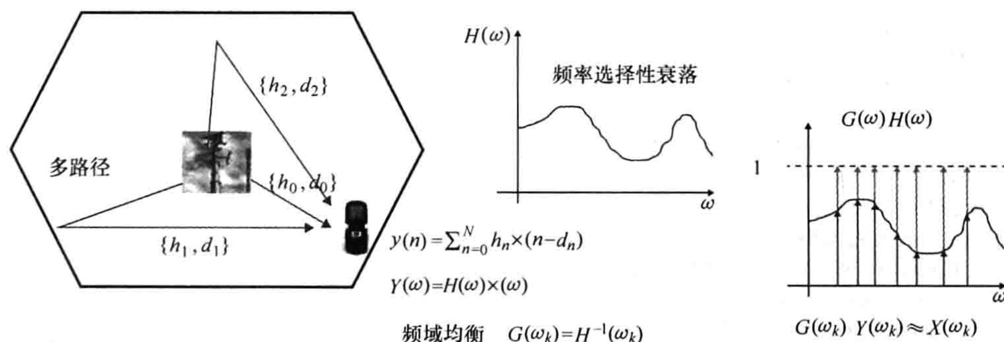


图 5.1 多径传播、频率选择性衰落，和频域均衡

5.1.3 多普勒效应

对移动系统基带传输，如 LTE，主信道劣化是由于多径传播引起短期衰落造成的。我们需要考虑衰落信道的效应以精确评估 LTE 系统性能。当移动终端移动时，信道冲击响应会随之变化。快和慢衰落信道反映了移动终端的速度，这就是多普勒相移的表现。

5.1.4 MATLAB 实例

我们可以使用多个通信系统工具箱的信道模型研究传输信号信道响应。Rayleigh 和 Rician 信道对象可以用来对单传输路径建模，而 comm. MIMOChannel 系

统对象可以用来研究多路天线和多径传播效应。所有这些组件都将延时和多普勒相移作为参数模拟信道衰落动态。

为了熟悉这些系统对象，让我们分别关注四种信道类型。他们之间的区别在于是否是频率平坦或选择性信道以及是否考虑接收端移动造成的多普勒相移及其引发的频率色散。

5.1.4.1 低移动率平坦衰落信道

第一个信道类型是低移动率平坦衰落信道。在这种情况下，延时特性并不包括多路时移。它由接收端和发射端时间差计算的延时值描述。不仅如此，低移动率造成的多普勒相移也近似为零。下面的 MATLAB 函数描述了如上信道。

Algorithm

MATLAB function

```
function y = ChanModelFading(in, Chan)
% Static (No mobility) Flat Fading Channel
%#codegen
% Get simulation params
numTx=1;

numRx=1;
chanSRate = Chan.chanSRate;
PathDelays = Chan.PathDelays;
PathGains  = Chan.PathGains;
Doppler      = Chan.DopplerShift;
% Initialize objects
persistent chanObj
if isempty(chanObj)
    chanObj = comm.MIMOChannel(...
        'SampleRate', chanSRate, ...
        'MaximumDopplerShift', Doppler, ...
        'PathDelays', PathDelays,...
        'AveragePathGains', PathGains,...
        'NumTransmitAntennas', numTx,...
        'TransmitCorrelationMatrix', eye(numTx),...
        'NumReceiveAntennas', numRx,...
        'ReceiveCorrelationMatrix', eye(numRx),...
        'PathGainsOutputPort', false,...
        'NormalizePathGains', true,...
        'NormalizeChannelOutputs', true);
end
y = step(chanObj, in);
```

为了直观看到此类信道的效应，我们向包括编码绕码和调制的系统中添加衰落信道并观察输入解调器的输入信号。注意通过运行下面这个 MATLAB 函数，我们可以观察到即使平坦衰落信道这样影响轻微的信道响应也会明显造成性能劣化。

Algorithm

MATLAB function

```
function [ber, bits] = chap5_ex01(EbNo, maxNumErrs, maxNumBits, prmLTE)
%#codegen
%% Constants
FRM=2432+24;
Kplus=FRM+24;
Indices = lteIntrlvrIndices(Kplus);
ModulationMode=1;
k=2*ModulationMode;
maxIter=6;
CodingRate=1/2;
snr = EbNo + 10*log10(k) + 10*log10(CodingRate);
noiseVar = 10.^(-snr/10);
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u = randi([0 1], FRM,1);                              % Randomly generated input bits
   data= CbCRCGenerator(u);                              % Transport block CRC code
   [t1, Kplus, C] = TbChannelCoding(data, prmLTE);
   t2 = Scrambler(t1, nS);                               % Scrambler
   t3 = Modulator(t2, ModulationMode);                   % Modulator
   % Channel & Add AWG noise
   [rxFade, ~] = MIMOFadingChan(t3, prmLTE);
   nVar = 10.^(0.1.*(-EbNo));                            % assume unit sigPower
   c0 = AWGNChannel2(rxFade, nVar );                     % AWGN channel
   % Receiver
   r0 = DemodulatorSoft(c0, ModulationMode, noiseVar);   % Demodulator
   r1 = DescramblerSoft(r0, nS);                         % Descrambler
   r2 = RateDematcher(r1, Kplus);                        % Rate Matcher
   r3 = TurboDecoder(-r2, Indices, maxIter);             % Turbo Decoder
   y = CbCRCDetector(r3);                                % Code block CRC detector
   % Measurements
   numErrs   = numErrs + sum(y~=u);                      % Update number of bit errors
   numBits   = numBits + FRM;                            % Update number of bits processed
   % Manage slot number with each subframe processed
   nS = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;                                   % Compute Bit Error Rate (BER)
```

图 5.2 显示了传输带宽内发射信号和接受信号的频率响应。它反映了平坦衰落信道的名称由来，在全带宽范围内每个频率都衰落了相同的值。

5.1.4.2 高移动率平坦衰落信道

我们现在对多普勒相移设置一个非零值以对高移动率平坦衰落信道建模。注意信道响应始终是平坦衰落，只是特定频率的增益为时间的函数。并且，接受信

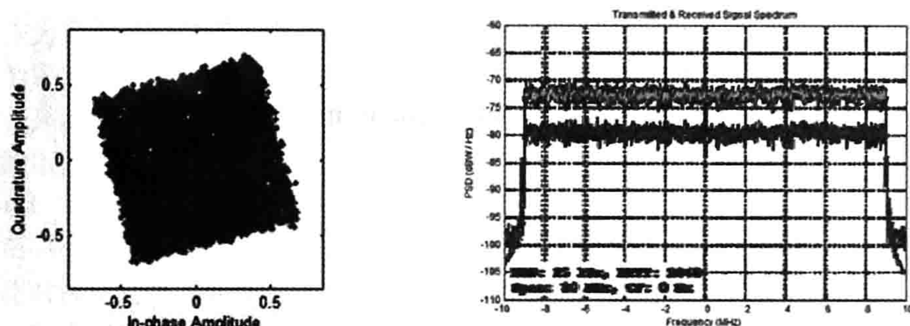


图 5.2 低移动率平坦衰落信道

号的星座图为 64QAM（正交幅度调制）调制。多普勒效应造成相位偏移使星座图在随时间旋转。这些效应显示在图 5.3 中。

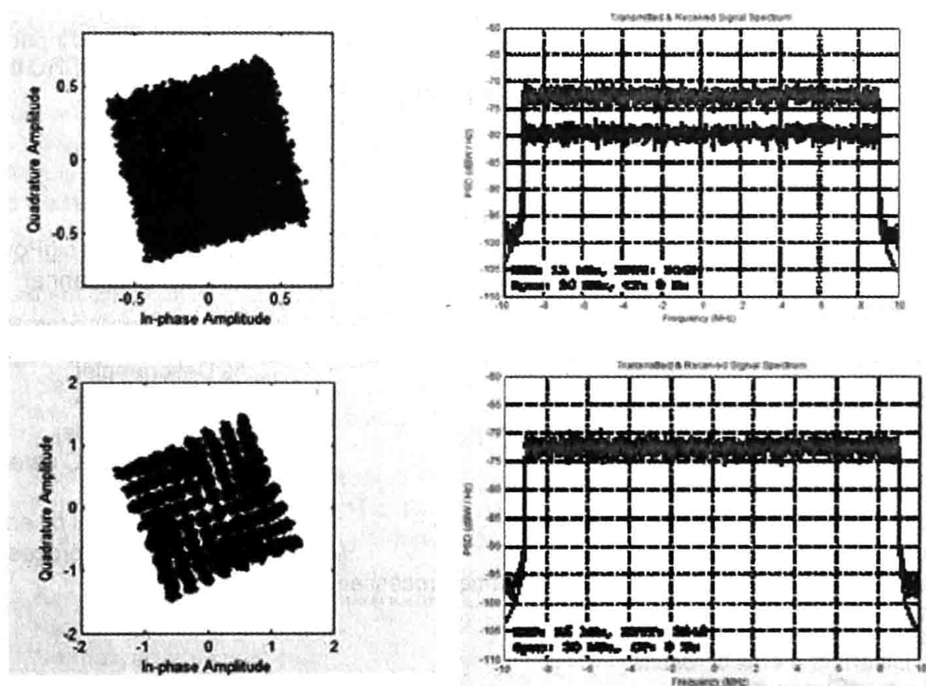


图 5.3 高移动率平坦衰落信道

5.1.4.3 低移动率频率选择性信道

在本节中，我们考察不包括多普勒相移但含有延时向量的频率选择性信道模型。通过时延向量，我们可以得到有相同值的增益向量。频率选择性信道相应如

图 5.4 所示。

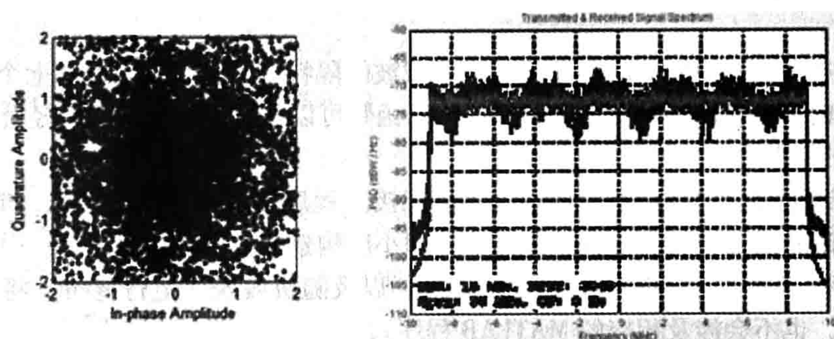


图 5.4 低移动率频率选择性衰落

5.1.4.4 高移动率频率选择性信道

最后，通过设定非零多普勒相移值我们可以对高移动率频率选择性信道建模。如同前面的高移动率情况，我们观察到信道增益随频率不同变化。我们也注意到信道相应随时间变化。图 5.5 表示了两个子帧 10ms 片段的信道相应频谱。

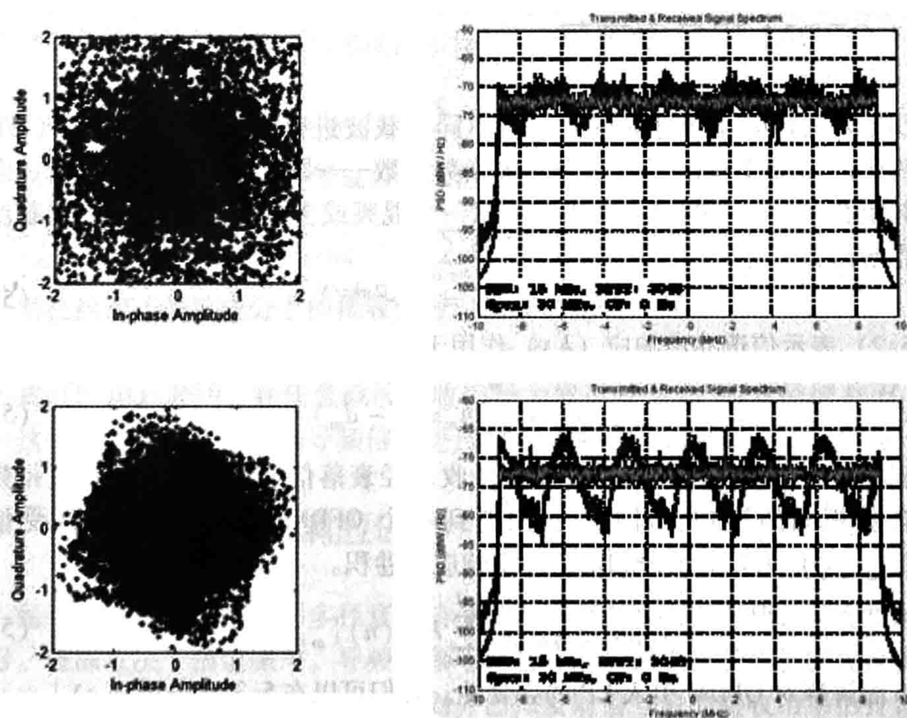


图 5.5 高移动率频率选择性衰落

5.2 讨论范围

在本书中，我们关注引导 15kHz 子载波间隔特定时间帧（每时隙七个 OFDM 符号）的普通循环前缀（CP）。MATLAB 函数可以通过改变几个参数轻松仿真扩展 CP 的情况。

在这个部分中我们不会详细讲述系统接入程序、启动、随机接入，或握手。我们讨论稳态信号处理下行链路传输，即小区构建完成下的一次通话。因此，同步信号和广播信道（BCHs）（下行链路）以及随机接入（上行链路）将不会重点讨论，也不会涉及相应的 MATLAB 程序。

5.3 工作流程

我们从编码、调制，和绕码开始，添加平坦和频率选择性信道模型。在本章中，我们讨论单天线传输（包括单输入单输出（SISO）和单输入多输出（SIMO））。我们关注参考信号生成，定义资源网格，和 OFDM 传输。最后，我们将下行链路传输的第一个系统模型的测试脚本集合在一起。

5.4 OFDM 和多径衰落

OFDM 调制信号是对资源元素在不同子载波进行反向傅里叶变换（IFFT）计算得到的。IFFT 输出可以认为是复指数函数——即基本函数如复正弦、混频或多频复合函数——的和。让我们考虑一个混频或多频复合（如一个子载波上的负指数函数）如：

$$x(n)\rfloor_{\omega=k\Delta f}=a_k\mathrm{e}^{\mathrm{j}2\pi kn/N} \tag{5.1}$$

式（5.2）表示信道冲击响应（h_m）作用于发射信号 $x(n)$后得到接收信号$y(n)$。

$$y(n)=\sum_{m=0}^{M}h_m x(n-d_m) \tag{5.2}$$

现在，由于成线性，当 OFDM 信号收多径衰落信道影响时，每个复指数成分也会受相同的影响。因此，我们得到每个 OFDM 子载波成分的接受信号（$y(n)\rfloor_{\omega=k\Delta f}$）为发射信号和信道冲击响应的卷积。

$$y(n)\rfloor_{\omega=k\Delta f}=\sum_{m=0}^{M}h_m x(n)\mid_{\omega=k\Delta f} \tag{5.3}$$

下面解释对 OFDM 引入 CP 的必要性。我们可以在 5.3 式中用 $x(n)\rfloor_{\omega=k\Delta f}=a_k\mathrm{e}^{\mathrm{j}2\pi kn/\overline{N}}$ 替代复指数成分。当且仅当多径延迟值超过 CP 范围时，OFDM 符号边

界重叠且子载波间正交性小时。现在，假设延迟在 CP 范围以内，我们可以得到接收信号子载波成分与发射信号子载波成分的关系：

$$y(n)\rfloor_{\omega=k\Delta f}=\sum_{m=0}^{M}h_m a_k \mathrm{e}^{\mathrm{j}2\pi k(n-d_m)/N} \tag{5.4}$$

通过一些代数运算，输出可以表示为

$$y(n)\rfloor_{\omega=k\Delta f}=a_k \mathrm{e}^{\frac{\mathrm{j}2\pi kn}{N}}\sum_{m=0}^{M}h_m \mathrm{e}^{-\frac{\mathrm{j}2\pi kd_m}{N}} \tag{5.5}$$

注意式子的最后一项，$\sum_{m=0}^{M}h_m \mathrm{e}^{-\frac{\mathrm{j}2\pi kd_m}{N}}$，这个增益项的复指数成分并不是时间 n 的函数但可以表示为子载波 k 的函数。通过定义增益项 $H_k=\sum_{m=0}^{M}h_m \mathrm{e}^{-\frac{\mathrm{j}2\pi kd_m}{N}}$ 并代入式（5.5），我们可以得到接收 OFDM 信号如下：

$$y(n)\rfloor_{\omega=k\Delta f}=H_k a_k \mathrm{e}^{\frac{\mathrm{j}2\pi kn}{N}} \tag{5.6}$$

现在我们看接收端的 OFDM 处理。移除 CP 之后，接收端首先进行傅里叶变换，如下式定义：

$$Y(\omega)=\sum_{n=0}^{N}y(n)\mathrm{e}^{-\mathrm{j}2\pi\omega n/N} \tag{5.7}$$

当我们对接受信号进行傅里叶变换，$y(n)=\frac{1}{N}\sum_{k=1}^{N}Y(\omega)\mathrm{e}^{\mathrm{j}2\pi kn/N}$，因 IFFT 基本函数的正交性，除了子载波成分之外的所有频率项内积全部消失。唯一一个决定接受信号傅里叶变换的非零项属于接收子载波成分，为

$$Y(\omega)\rfloor_{\omega=k\Delta f}=\frac{1}{N}\sum_{n=0}^{N}y(n)\rfloor_{\omega=k\Delta f}\mathrm{e}^{-\mathrm{j}2\pi\omega_k n/N} \tag{5.8}$$

通过代入接收 OFDM 信号成分，我们得到下式：

$$Y(\omega)\rfloor_{\omega=k\Delta f}=\frac{1}{N}\sum_{n=0}^{N}H_k a_k \mathrm{e}^{\mathrm{j}2\pi k\Delta fn/N}\mathrm{e}^{-\frac{\mathrm{j}2\pi k\Delta fn}{N}} \tag{5.9}$$

对在给定子载波成分上的接收信号，将该式化简为更直观的形式：

$$Y(\omega)\rfloor_{\omega=k\Delta f}=H_k a_k \tag{5.10}$$

式（5.10）表明，在任意载波接收信号为发射符号 a_k 和多径增益 H_k 的乘积。这个简单的结果是使用导频信号进行频域均衡的定义基础。

5.5 OFDM 和信道响应估计

每个发射信号成分受到多径衰落信道影响都会在接收端得到一个有损的接受信号。衰减取决于信道噪声。导频或参考信号可以认为是在子载波固定位置存在的信号。我们可以在这些子载波通过划分已知发射信号值的接收结果估计信道响应。每个特定子载波的信道响应可以表示为

$$H(\omega)\rfloor_{\omega=k\Delta f}=\frac{Y(\omega)\rfloor_{\omega=k\Delta f}}{X(\omega)\rfloor_{\omega=k\Delta f}}$$

$$H(\omega)\rfloor_{\omega=k\Delta f}=\frac{H_k a_k}{a_k}$$

$$H(\omega)\rfloor_{\omega=k\Delta f}=H_k \tag{5.11}$$

通过若干形式的插值逼近，我们现在可以估计所有子载波的信道响应，不仅仅是已知子载波。这可以让我们在频域对均衡定义减小衰落信道效应。它比传统的通过估计信道冲击响应和适应性滤波器均衡接受信号的时域均衡技术更有效率。

5.6 频域均衡

OFDM 最重要的特性之一就是其对多径衰落的稳健性和有效应对多径衰落。OFDM 通过频域均衡对衰落效应补偿。不同于在时域对接受信号进行滤波以及对信道冲击响应求反，OFDM 首先进行一个频域数据变换然后使用参考信号对信道频率响应求反。

这一过程可分为两个步骤。首先，建立时 - 频资源网格结构，在时域生成 OFDM 符号之前，数据在频域上映射到子载波。这一步即资源元素映射。构成 LTE 下行链路资源网格的信号类型如下：

1）用户数据（物理下行公共信道，PDSCH）；

2）小区特有参考（CSR）信号（也就是导频信号）；

3）主同步信号（PSS）和辅助同步信号（SSS）；

4）物理广播信道（PBSCH）；

5）物理下行链路控制信道（PDCCH）。

第二步，我们取资源元素向量作为输入并生成 OFDM 符号。这一处理包括使用 IFFT 操作生成 OFDM 调制信号和插入 CP。使用 CP 可以使接收端在一个精确的时域周期采样每个 OFDM 符号。当信道延迟小于 CP 长度时，CP 可以帮助减轻码间串扰。

生成 OFDM 信号之前，我们需要以类型 -1 和类型 -2 帧结构生成资源网格。因为我们在本书中讲解频分双工，故我们会在这里使用类型 1 帧结构。下面我们将说明如何生成构成资源网格的相关信号以及 OFDM 符号。

5.7 LTE 资源网格

理解数据时 - 频映射，组织资源网格，是理解 LTE 传输方案的关键。资源

网格本质上是一个由调制映射器生成调制符号组成的矩阵。在2D映射下，网格的 y 轴表示子载波对应的频域而 x 轴表示 OFDM 符号对应的时域[2]。

资源网格内数据的位置十分重要，体现一些 LTE 物理层模型的设计参数。比如，导频信号（CSR）延资源网格两轴的位置和解析度确定了时域和频域信道响应估计的精度。同样，PDSCH 控制信道信息被安排到每个子载波的初始位置，帮助接收端在开始译码用户数据之前解码重要处理参数（如使用的调制类型和 MIMO 模式）。

安排数据在资源网格的位置可以单纯理解为组织时间帧顺序和按照 LTE 定义帧、子帧，和时隙。每个 LTE 子帧为 10ms，它由十个 1ms 子帧构成并编号为从 0 到 9。每个子帧可以分为两个 0.5ms 的时隙，每个时隙在使用普通 CP 情况下包括 7 个 OFDM 符号，在使用扩展 CP 情况下为 6 个 OFDM 符号。

资源网格中每种调制数据类型的位置（用户数据，CSR，DCI，PSS，SSS，和 BCH）在时域和频域遵循特定结构。这个结构和三个参数有关：子载波（y 轴）指数，OFDM 符号（x 轴）指数，和 10ms 帧内 1ms 子帧的指数。一个帧中所有子帧包括三种数据类型：用户数据（PDSCH），导频 CSR，和下行链路控制数据（PDCCH）。PSS 和 SSS 只在特定子载波索引（围绕资源网格中心的 72 个子载波）和子帧 0 和 5 上特定 OFDM 符号索引（SSS 在第五个符号而 PSS 在第六个符号）有效。PBCH 只位于特定子载波索引（围绕资源网格中心的 72 个子载波）和子载波 0 特定 OFDM 符号索引（从第七个到第十个符号）上。图 5.6 表示了不同信号类型调制数据在资源网格上的位置。

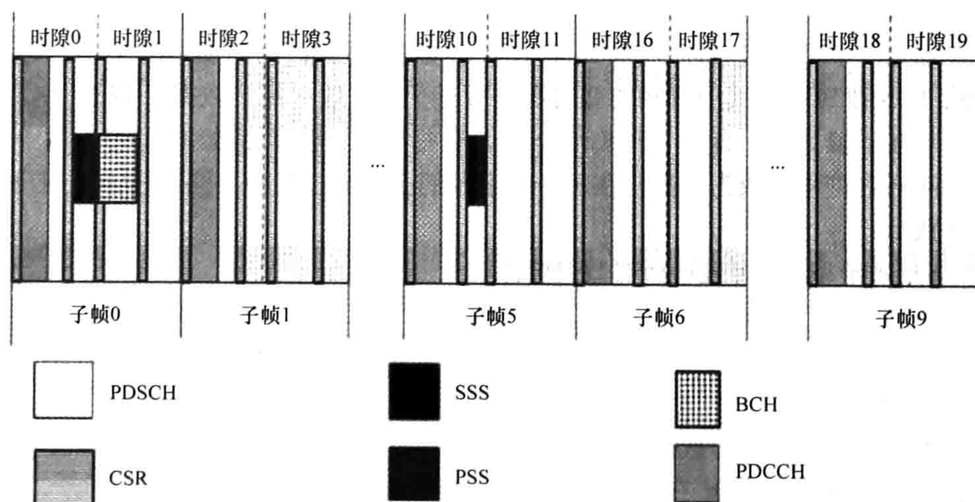


图 5.6 LTE 资源网格内容——全网格图——包括六种信号类型

5.8 配置资源网格

让我讨论资源网格大小和如何更新每个子帧。在本书中，我们假设 LTE 收发端（发送端、信道模型，和接收端）同时处理一个子帧。因每个子帧长度为 1ms，处理一秒的数据需要 1000 个这样的收发端。

在每个子帧上，资源网格大小（N_{total} = 占满网格的全部符号数量）和下面四个参数有关：

$$\begin{cases} N_{\mathrm{rb}} & \text{资源网格上资源块数} \\ N_{\mathrm{sc}} & \text{资源块上子载波数} \\ N_{\mathrm{sym}} & \text{时隙上符号数} \\ N_{\mathrm{slot}} & \text{子帧上时隙数} \end{cases}$$

整体资源网格大小是行（全部子载波数）和列（每个子帧上全部 OFDM 符号数）的乘积。全部子载波数是全部资源块（N_{rb}）和每个资源块上全部子载波数（N_{sc}）的乘积。每个子帧上全部 OFDM 符号数为每个时隙内符号数（N_{sym}）和一个子帧内时隙数（N_{slot}）的乘积。

$$N_{\mathrm{total}} = N_{\mathrm{rb}} \cdot N_{\mathrm{sc}} \times N_{\mathrm{sym}} \cdot N_{\mathrm{slot}} \tag{5.12}$$

每个子帧内时隙数（N_{slot}）恒为 2。每个时隙内符号数（N_{sym}）与 CP 类型有关。在本书中，我们使用普通 CP，所以一个时隙内符号数为 7。每个资源块上全部子载波数（N_{sc}）也与 CP 类型有关；当我们使用普通 CP 时，其值为 12。因此，资源网格大小完全由资源块数量决定，它直接与带宽相关。

如上一节所讨论的，资源元素有 6 种数据资源：用户数据、CSR、DCI、PSS，SSS 和 BCH。其中有些类型资源在所有子帧有效（用户数据，CSR，DCI），有些只在子帧 0 和 5 有效（PSS 和 SSS），有些只在子帧 0 有效（BCH）。因一个资源网格内符号总数为常数，在每个帧上我们必须通过三种不同途径计算用户数据的数量：

1）对子帧 0：可表示所有数据资源

2）对子帧 5：除了用户数据，还可表示 CSR、DCI、PSS，和 SSS

3）其他帧 {1，2，3，4，6，7，8，9}：除了用户数据，只可以表示 CSR 和 DCI 符号

图 5.7 描述了 6 种数据类型在资源网格的相应位置并关注网格中心 6 个资源块，PSS、SSS，和 BCH 在其子帧有效。

5.8.1 CSR 符号

另外，CSR 以特定时频图图形存在于每个子帧的每个资源块。在单天线配

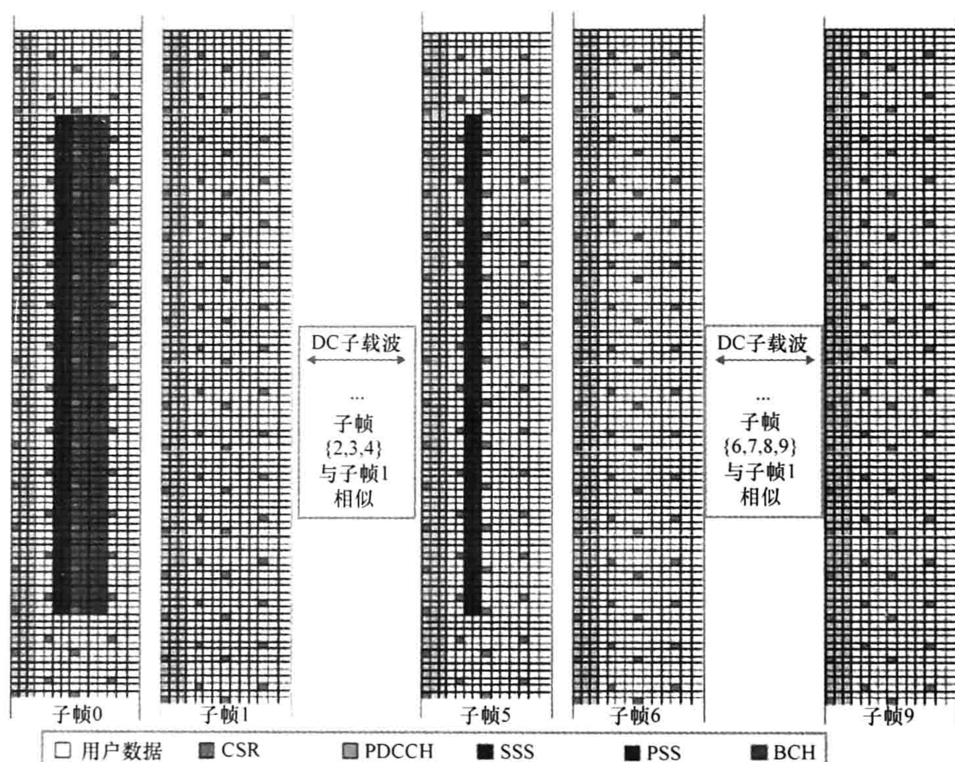


图 5.7 LTE 资源网格——关注中心附近（DC 子载波）6 个资源块——包括 6 种数据类型

置情况下，LTE 定义每个子帧上四个 OFDM 符号 {0，5，7，12} 的资源块有两个 CSR 符号。在 OFDM 符号 0 和 7 上，起始索引符为第一个子载波，而在符号 5 和 12 上起始索引符为第 5 个子载波。两个 CSR 子载波被 6 个子载波分开。一个资源网格共有 $N_{csr}=8N_{rb}$ 个 CSR 符号可用。

5.8.2 DCI 符号

DCI 位于每个子帧第 N 个 OFDM 符号上，N 为 1，2 或 3。DCI 携带了 PDSCH、PCFICH（物理控制格式索引信道），和 PHICH（物理混合 ARQ 索引信道）内容，它们在分布于每个子帧上第 1 个 OFDM 符号的 CSR 数据之外，占据了第一个甚至第二个和第三个 OFDM 符号的全部资源元素。每个子帧的 DCI 大小为 $N_{DCI}=N_{rb}(10+12(N-1))$。在本章中我们不在资源网格中放入 DCI；我们将会在第 7 章详细讨论 DCI。

5.8.3 BCH 符号

PBCH 位于子帧 0 并占据从第 7 到第 10 个 OFDM 符号的 6 个中心资源块。

因第 7 个 OFDM 符号包含 CSR 符号，其 BCH 大小只有 60（72 - 2 × 6），而在之后的三个符号中 BCH 大小为 72。所有帧的 BCH 大小总和为 $N_{\text{BCH}} = 60 + 3 \times 72 = 276$。

5.8.4 同步符号

PSS 和 SSS 位于以 DC 子载波为中心的 6 个资源块上。在子帧 0 和 5，PSS 占据第 6 个 OFDM 符号而 SSS 占据第 5 个资源符号。因在这两个符号上没有 CSR 信号，每个子帧的总同步信号大小为 $N_{\text{PSS}} = N_{\text{SSS}} = 72$，因一个帧包括两个子帧，因此一个帧的同步信号大小为 144。

5.8.5 用户数据符号

资源网格内数据总量与资源块数量或本质上与带宽有关。资源元素有 6 种数据资源类型（用户数据、CSR、DCI、PSS，SSS 和 BCH）。因此，假如单位带宽恒定，则资源网格大小可表示为

$$N_{\text{total}} = N_{\text{user data}} + N_{\text{CSR}} + N_{\text{DCI}} + N_{\text{PSS}} + N_{\text{SSS}} + N_{\text{BCH}} \tag{5.13}$$

BCH 或同步信号的存在与否取决于子帧索引。因此，在子帧上的用户数据大小也取决于子帧索引，可如下表示：

1）对子帧 0，可表示所有数据资源：

$$N_{\text{user data}} = N_{\text{total}} - (N_{\text{CSR}} + N_{\text{DCI}} + N_{\text{PSS}} + N_{\text{SSS}} + N_{\text{BCH}}) \tag{5.14}$$

2）对子帧 5，除了用户数据，还可表示 CSR、DCI、PSS，和 SSS：

$$N_{\text{user data}} = N_{\text{total}} - (N_{\text{CSR}} + N_{\text{DCI}} + N_{\text{PSS}} + N_{\text{SSS}}) \tag{5.15}$$

3）其他帧 {1，2，3，4，6，7，8，9}，除了用户数据，只可以表示 CSR 和 DCI 符号：

$$N_{\text{user data}} = N_{\text{total}} - (N_{\text{CSR}} + N_{\text{DCI}}) \tag{5.16}$$

下面的 MATLAB 函数重点进行如上计算并设置了一些 PDSCH 参数。函数取 3 个参数作为输入声明：信道带宽（chanBW），在每个子帧上对应控制信道的 OFDM 符号数（contReg），和调制类型（modType）。函数计算了 PDSCH 处理过程中的很多参数，包括详细的资源网格。

Algorithm

MATLAB function

```
function p= prmsPDSCH(chanBW, contReg, modType, varargin)
% Returns parameter structures for LTE PDSCH simulation.
%
% Assumes a FDD, normal cyclic prefix, full-bandwidth, single-user
% SISO or SIMO downlink transmission.
%% PDSCH parameters
```

```
switch chanBW
   case 1 % 1.4 MHz
      BW = 1.4e6; N = 128; cpLen0 = 10; cpLenR = 9;
      Nrb = 6; chanSRate = 1.92e6;
   case 2 % 3 MHz
      BW = 3e6; N = 256; cpLen0 = 20; cpLenR = 18;
      Nrb = 15; chanSRate = 3.84e6;
   case 3 % 5 MHz
      BW = 5e6; N = 512; cpLen0 = 40; cpLenR = 36;
      Nrb = 25; chanSRate = 7.68e6;
   case 4 % 10 MHz
      BW = 10e6; N = 1024; cpLen0 = 80; cpLenR = 72;
      Nrb = 50; chanSRate = 15.36e6;
   case 5 % 15 MHz
      BW = 15e6; N = 1536; cpLen0 = 120; cpLenR = 108;
      Nrb = 75; chanSRate = 23.04e6;
   case 6 % 20 MHz
      BW = 20e6; N = 2048; cpLen0 = 160; cpLenR = 144;
      Nrb = 100; chanSRate = 30.72e6;
end
p.BW = BW;                  % Channel bandwidth
p.N = N;                % NFFT
p.cpLen0 = cpLen0;          % Cyclic prefix length for 1st symbol
p.cpLenR = cpLenR;           % Cyclic prefix length for remaining
p.Nrb = Nrb;               % Number of resource blocks
p.chanSRate = chanSRate;    % Channel sampling rate
p.contReg = contReg;
if nargin > 3, numTx=varargin{4};else numTx=1;end
if nargin > 4, numRx=varargin{5};else numRx=1;end
p.numTx = numTx;
p.numRx = numRx;
p.numLayers = 1;
p.numCodeWords = 1;
% For Normal cyclic prefix, FDD mode
p.deltaF = 15e3;   % subcarrier spacing
p.Nrb_sc = 12;     % no. of subcarriers per resource block
p.Ndl_symb = 7;    % no. of OFDM symbols in a slot
% Actual PDSCH bits calculation - accounting for PDCCH, PBCH, PSS, SSS
numResources = (p.Nrb*p.Nrb_sc)*(p.Ndl_symb*2);
numCSRRE = 2*2*2 * p.Nrb;           % CSR, RE per OFDMsym/slot/subframe per RB
numContRE = (10 + 12*(p.contReg-1))*p.Nrb;
numBCHRE = 60+72+72+72;             % removing the CSR present in 1st symbol
numSSSRE=72;
numPSSRE=72;
numDataRE=zeros(3,1);
% Account for BCH, PSS, SSS and PDCCH for subframe 0
numDataRE(1)=numResources-numCSRRE-numContRE-numSSSRE - numPSSRE-
numBCHRE;
% Account for PSS, SSS and PDCCH for subframe 5
```

```
numDataRE(2)=numResources-numCSRRE-numContRE-numSSSRE - numPSSRE;
% Account for PDCCH only in all other subframes
numDataRE(3)=numResources-numCSRRE-numContRE;
% Maximum data resources - with no extra overheads (only CSR + data)
p.numResources=numResources;
p.numCSRResources =  numCSRRE;
p.numContRE = numContRE;
p.numBCHRE = numBCHRE;
p.numSSSRE=numSSSRE;
p.numPSSRE=numPSSRE;
p.numDataRE=numDataRE;
p.numDataResources = p.numResources - p.numCSRResources;
% Modulation types , bits per symbol, number of layers per codeword
Qm = 2 * modType;
p.Qm = Qm;
p.numLayPerCW = p.numLayers/p.numCodeWords;
% Maximum data bits - with no extra overheads (only CSR + data)
p.numDataBits = p.numDataResources*Qm*p.numLayPerCW;
numPDSCHBits =numDataRE*Qm*p.numLayPerCW;
p.numPDSCHBits = numPDSCHBits;
p.maxG = max(numPDSCHBits);
```

在本章中我们不讨论 DCI、BCH 和同步信号生成。我们关注计算 CSR 的内容和用户数据覆盖资源网格和使用传输模式 1 进行 OFDM 传输。

5.9 参考信号生成

为了确保发送端和接收端生成相同的 CSR 参考序列，LTE 定义 Gold 序列并在收发端出端初始化参数。这些参数包括小区身份数（NcellID）、子帧索引（nS）、时隙索引（i），和时隙内包含 CSR 的 OFDM 符号索引（IIdx）。

函数包括两个输入声明：子帧索引（nS）和发射天线数（numTx）。因我们在本章中只对单天线情况进行建模，参数发射天线数为 1。为了方便起见，我们在随后章节针对多天线也使用同一个函数。基于 Gold 序列和所有可用天线接口，函数生成 CSR 数以供信道估计所需。输出变量 y 为大小等于行列乘积的矩阵。行数为资源网格内最大 CSR 数，列数为发射天线数。以下 MATLAB 函数表示了每个 CSR 信号元素是如何生成的。

Algorithm

MATLAB function

```
function y = CSRgenerator(nS, numTx)
% LTE Cell-Specific Reference signal generation.
%  Section 6.10.1 of 3GPP TS 36.211 v10.0.0.
%  Generate the whole set per OFDM symbol, for 2 OFDM symbols per slot,
%  for 2 slots per subframe, per antenna port (numTx).
%  This fcn accounts for the per antenna port sequence generation, while
%  the actual mapping to resource elements is done in the Resource mapper.
%#codegen
persistent hSeqGen;
persistent hInt2Bit;
% Assumed parameters
NcellID = 0;        % One of possible 504 values
Ncp = 1;            % for normal CP, or 0 for Extended CP
NmaxDL_RB = 100;    % largest downlink bandwidth configuration, in resource blocks
y = complex(zeros(NmaxDL_RB*2, 2, 2, numTx));
l = [0; 4];    % OFDM symbol idx in a slot for common first antenna port
% Buffer for sequence per OFDM symbol
seq = zeros(size(y,1)*2, 1); % *2 for complex outputs
if isempty(hSeqGen)
   hSeqGen = comm.GoldSequence('FirstPolynomial',[1 zeros(1, 27) 1 0 0 1],...
                    'FirstInitialConditions', [zeros(1, 30) 1], ...
                    'SecondPolynomial', [1 zeros(1, 27) 1 1 1 1],...
                    'SecondInitialConditionsSource', 'Input port',...
                    'Shift', 1600,...
                    'SamplesPerFrame', length(seq));
   hInt2Bit = comm.IntegerToBit('BitsPerInteger', 31);
end
% Generate the common first antenna port sequences
for i = 1:2 % slot wise
   for lIdx = 1:2 % symbol wise
      c_init = (2^10)*(7*((nS+i-1)+1)+l(lIdx)+1)*(2*NcellID+1) + 2*NcellID + Ncp;
      % Convert to binary vector
      iniStates = step(hInt2Bit, c_init);
      % Scrambling sequence - as per Section 7.2, 36.211
      seq = step(hSeqGen, iniStates);
      % Store the common first antenna port sequences
      y(:, lIdx, i, 1) = (1/sqrt(2))*complex(1-2.*seq(1:2:end), 1-2.*seq(2:2:end));
   end
end
% Copy the duplicate set for second antenna port, if exists
if (numTx>1)
   y(:, :, :, 2) = y(:, :, :, 1);
end
```

```
% Also generate the sequence for l=1 index for numTx = 4
if (numTx>2)
   for i = 1:2 % slot wise
      % l = 1
      c_init = (2^10)*(7*((nS+i-1)+1)+1+1)*(2*NcellID+1) + 2*NcellID + Ncp;
      % Convert to binary vector
      iniStates = step(hInt2Bit, c_init);
      % Scrambling sequence - as per Section 7.2, 36.211
      seq = step(hSeqGen, iniStates);
      % Store the third antenna port sequences
      y(:, 1, i, 3) = (1/sqrt(2))*complex(1-2.*seq(1:2:end), 1-2.*seq(2:2:end));
   end
   % Copy the duplicate set for fourth antenna port
   y(:, 1, :, 4) = y(:, 1, :, 3);
end
```

5.10 资源元素映射

在本节中，我们详细考察资源网格按标准制定位置放置资源元素的资源元素映射过程。映射本质上是生成一个资源网格矩阵索引符并将多种信息类型放置入网格的过程。三种不同类型的资源块表示在图 5.8 ~ 图 5.10 中帮助直观认识三种索引。根据使用的子帧不同，我们在围绕资源网格中心 DC 子载波的 6 个资源块的子帧 0 和子帧 5 上放置 BCH、PSS，和 SSS。CSR 放置在每个时隙的符号 0 和 5 上，它们在频域间隔六个子载波。

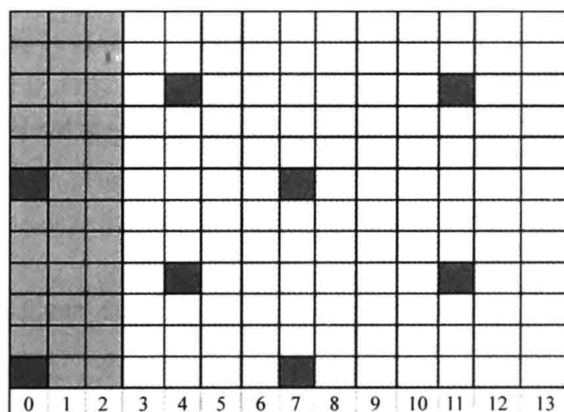


图 5.8　资源元素映射：子帧 1，2，3，4，5，7，8 和 9 + 的所有资源块以及子帧 0 和 5 的非中心资源块。包括 DCI、CSR，和用户数据

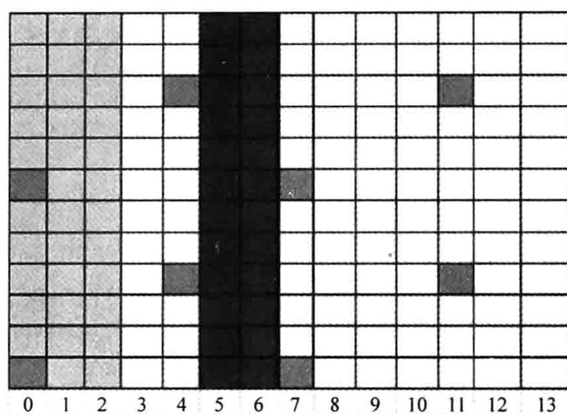


图 5.9 资源元素映射：子帧 5 的中心资源块，
包括 PSS、SSS、DCI、CSR 和用户数据

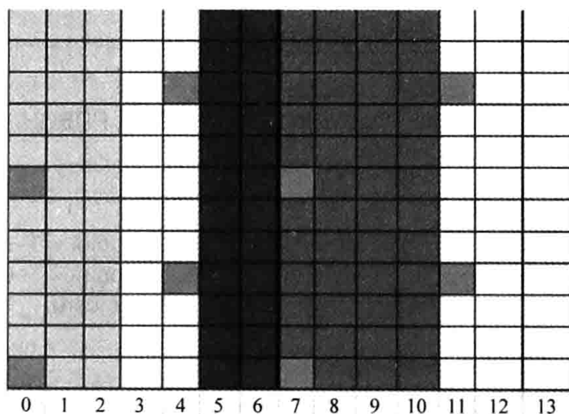


图 5.10 资源元素映射：子帧 0 的中心资源块，
包括 BCH、PSS、SSS、DCI、CSR 和用户数据

下面的 MATLAB 函数表示了资源元素映射。因为 MATLAB 使用 1 - 索引计数法，我们在矩阵中对不同资源元素生成从 1 开始的索引而不是从 0 开始。函数取用户数据（in）、CSR 信号（csr）、子帧索引（nS），和描述结构的被称为 preLTE 的 PDSCH 参数作为输入。根据 BCH、SSS、PSS 和 DCI 的不同，函数可引入更多输入。输出变量 y 为资源网格矩阵。2D 网格矩阵的行数为子载波数，列数共有 14 列（2 个时隙每个包含 7 个 OFDM 符号）。

Algorithm

MATLAB function

```
function y = REmapper_1Tx(in, csr, nS, prmLTE, varargin)
%#codegen
switch nargin
   case 4, pdcch=[];pss=[];sss=[];bch=[];
   case 5, pdcch=varargin{1};pss=[];sss=[];bch=[];
   case 6, pdcch=varargin{1};pss=varargin{2};sss=[];bch=[];
   case 7, pdcch=varargin{1};pss=varargin{2};sss=varargin{3};bch=[];
   case 8, pdcch=varargin{1};pss=varargin{2};sss=varargin{3};bch=varargin{4};
   otherwise
      error('REMapper has 4 to 8 arguments!');
end
% NcellID = 0;                          % One of possible 504 values
% numTx = 1;                            % prmLTE.numTx;
% Get input params
Nrb = prmLTE.Nrb;                       % either of {6, }
Nrb_sc = prmLTE.Nrb_sc;                 % 12 for normal mode
Ndl_symb = prmLTE.Ndl_symb;             % 7    for normal mode
numContSymb    = prmLTE.contReg;  % either {1, 2, 3}
% Initialize output buffer
y = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2));
%% Specify resource grid location indices for CSR, PDCCH, PDSCH, PBCH, PSS, SSS
%% 1st: Indices for CSR pilot symbols
lenOFDM = Nrb*Nrb_sc;
idx         = 1:lenOFDM;
idx_csr0    = 1:6:lenOFDM;          % More general starting point = 1+mod(NcellID, 6);
idx_csr4    = 4:6:lenOFDM;          % More general starting point = 1+mod(3+NcellID, 6);
idx_csr     =[idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0,
11*lenOFDM+idx_csr4];
%% 2nd: Indices for PDCCH control data symbols
ContREs=numContSymb*lenOFDM;
idx_dci=1:ContREs;
idx_pdcch = ExpungeFrom(idx_dci,idx_csr0);
%% 3rd: Indices for PDSCH and PDSCH data in OFDM symbols where pilots
are present
idx_data0= ExpungeFrom(idx,idx_csr0);
idx_data4 = ExpungeFrom(idx,idx_csr4);
%% Handle 3 types of subframes differently
switch nS
   %% 4th: Indices for BCH, PSS, SSS are only found in specific subframes 0 and 5
   % These symbols share the same 6 center sub-carrier locations (idx_ctr)
   % and differ in OFDM symbol number.
   case 0 % Subframe 0
      % PBCH, PSS, SSS are available + CSR, PDCCH, PDSCH
      idx_6rbs = (1:72);
```

```
    idx_ctr = 0.5* lenOFDM - 36 + idx_6rbs ;
    idx_SSS  = 5* lenOFDM + idx_ctr;
    idx_PSS  = 6* lenOFDM + idx_ctr;
    idx_ctr0 = ExpungeFrom(idx_ctr,idx_csr0);
    idx_bch=[7*lenOFDM + idx_ctr0, 8*lenOFDM + idx_ctr, 9*lenOFDM + idx_ctr,
10*lenOFDM + idx_ctr];
    idx_data5  = ExpungeFrom(idx,idx_ctr);
    idx_data7 = ExpungeFrom(idx_data0,idx_ctr);
    idx_data  = [ContREs+1:4*lenOFDM,  4*lenOFDM+idx_data4, ...
      5*lenOFDM+idx_data5, 6*lenOFDM+idx_data5,  7*lenOFDM+idx_data7,
8*lenOFDM+idx_data5, ...
      9*lenOFDM+idx_data5, 10*lenOFDM+idx_data5, 11*lenOFDM+idx_data4, ...
      12*lenOFDM+1:14*lenOFDM];
    y(idx_csr)=csr(:);          % Insert Cell-Specific Reference signal (CSR) = pilots
    y(idx_data)=in;             % Insert Physical Downlink Shared Channel
(PDSCH) = user data
    if ~isempty(pdcch), y(idx_pdcch)=pdcch;end
% Insert Physical Downlink Control Channel (PDCCH)
    if ~isempty(pss), y(idx_PSS)=pss;end % Insert Primary Synchronization
Signal (PSS)
    if ~isempty(sss), y(idx_SSS)=sss;end
% Insert Secondary Synchronization Signal (SSS)
    if ~isempty(bch), y(idx_bch)=bch;end % Insert Broadcast Channel data (BCH)

  case 10 % Subframe 5
    % PSS, SSS are available + CSR, PDCCH, PDSCH
    % Primary and Secondary synchronization signals in OFDM symbols 5 and 6
    idx_6rbs = (1:72);
    idx_ctr = 0.5* lenOFDM - 36 + idx_6rbs ;
    idx_SSS  = 5* lenOFDM + idx_ctr;
    idx_PSS  = 6* lenOFDM + idx_ctr;
    idx_data5 = ExpungeFrom(idx,idx_ctr);
    idx_data  = [ContREs+1:4*lenOFDM, 4*lenOFDM+idx_data4,
5*lenOFDM+idx_data5, 6*lenOFDM+idx_data5, ...
      7*lenOFDM+idx_data0, 8*lenOFDM+1:11*lenOFDM,  11*lenOFDM+idx_data4, ...
      12*lenOFDM+1:14*lenOFDM];
    y(idx_csr)=csr(:);          % Insert Cell-Specific Reference signal (CSR) = pilots
    y(idx_data)=in;             % Insert Physical Downlink Shared Channel
(PDSCH) = user data
    if ~isempty(pdcch), y(idx_pdcch)=pdcch;end
% Insert Physical Downlink Control Channel (PDCCH)
    if ~isempty(pss), y(idx_PSS)=pss;end % Insert Primary Synchronization Signal (PSS)
    if ~isempty(sss), y(idx_SSS)=sss;end
% Insert Secondary Synchronization Signal (SSS)

  otherwise % other subframes
    % Only CSR, PDCCH, PDSCH
    idx_data = [ContREs+1:4*lenOFDM, 4*lenOFDM+idx_data4, ...
      5*lenOFDM+1:7*lenOFDM, ...
```

```
        7*lenOFDM+idx_data0, ...
        8*lenOFDM+1:11*lenOFDM, ...
        11*lenOFDM+idx_data4, ...
        12*lenOFDM+1:14*lenOFDM];
    y(idx_csr)=csr(:);          % Insert Cell-Specific Reference signal (CSR) = pilots
    y(idx_data)=in;             % Insert Physical Downlink Shared Channel
(PDSCH) = user data
    if ~isempty(pdcch), y(idx_pdcch)=pdcch;end
% Insert Physical Downlink Control Channel (PDCCH)
end
end
```

5.11 OFDM 信号生成

OFDM 信号生成在资源网格上进行。这个处理将 OFDM 符号一个接一个进行 IFFT 并附加 CP 生成 OFDM 调制信号。下面的 MATLAB 函数展示了在 IFFT 之前，数据是如何压入 FFT 缓存并重新排序去除 DC 子载波的。IFFT 之后我们对输出进行缩放。添加 CP 的过程是将 IFFT 输出最后 N 位采样添加到缓存开始。第一个 OFDM 符号的 N 值不同于其他所有 OFDM 符号。函数的输入为资源网格（in）和包含 PDSCH 参数的结构（prmLTE）。一个时隙内各个符号的 CP 长度各不相同。每个时隙的第一个 OFDM 符号的 CP 长度（cpLen0）略长于其他 6 个符号的 CP 值（cpLenR）。当计算输出信号，CP 的不同会体现在计算输出信号的 For 循环内，这个循环将每一个 OFDM 调制信号的长度顺序添加到每一个子帧内输出向量[3]。

函数的输出是一个 2D 矩阵：一维输出长度为每个子帧的输出，二维输出长度为天线端口数。因本章我们关注单天线配置，输出为列向量即二维输出长度为 1。我们不需要在下一张介绍 MIMO 时重写这个函数，因为它在单通道和多通道 OFDM 的情况下一样适用。

Algorithm

MATLAB function

```
function y = OFDMTx(in, prmLTE)
%#codegen
persistent hIFFT;
if isempty(hIFFT)
  hIFFT = dsp.IFFT;
end
```

```
[len, numSymb, numLayers] = size(in);
% N assumes 15KHz subcarrier spacing
N = prmLTE.N;
cpLen0 = prmLTE.cpLen0;
cpLenR = prmLTE.cpLenR;
slotLen = (N*7 + cpLen0 + cpLenR*6);
subframeLen = slotLen*2;
tmp = complex(zeros(N, numSymb, numLayers));
% Pack data, add DC, and reorder
tmp(N/2-len/2+1:N/2, :, :) = in(1:len/2, :, :);
tmp(N/2+2:N/2+1+len/2, :, :) = in(len/2+1:len, :, :);
tmp = [tmp(N/2+1:N, :, :); tmp(1:N/2, :, :)];
% IFFT processing
x = step(hIFFT, tmp);
x = x.*(N/sqrt(len));
% Add cyclic prefix per OFDM symbol per antenna port
% and serialize over the subframe (equal to 2 slots)
% For a subframe of data
y = complex(zeros(subframeLen, numLayers));
for j = 1:2 % Over the two slots
   % First OFDM symbol
   y((j-1)*slotLen+(1:cpLen0), :) = x((N-cpLen0+1):N, (j-1)*7+1, :);
   y((j-1)*slotLen+cpLen0+(1:N), :) = x(1:N, (j-1)*7+1, :);

   % Next 6 OFDM symbols
   for k = 1:6
      y((j-1)*slotLen+cpLen0+k*N+(k-1)*cpLenR+(1:cpLenR), :) = x(N-cpLenR+1:N,
(j-1)*7+k+1, :);
      y((j-1)*slotLen+cpLen0+k*N+k*cpLenR+(1:N), :) = x(1:N, (j-1)*7+k+1, :);
   end
end
```

5.12 信道建模

下面的 MATLAB 函数表示了 OFDM 信号的信道建模过程。这个函数来源于在本章前面的部分里我们建立的 SISO 信道模型。函数的输出为 OFDM 信号（in），包含 PDSCH 参数的结构（prmLTE），和流年干一个包含信道模型参数的结构（prmMdl）。基于这些输入参数，函数建立平坦或频率选择性信道。信道模型的输出（y）为接收端接受的信号。另一个输出（yPg）为一个包含衰落过程中信道路径增益的矩阵。这个信号可以用于估计“理想”信道相应。我们将会在下面的章节详细讲解理想信道相应并比较响应估计。

Algorithm

MATLAB function

```
function [y, yPg] = MIMOFadingChan(in, prmLTE, prmMdl)
% MIMOFadingChan
%#codegen
% Get simulation params
numTx        = prmLTE.numTx;
numRx        = prmLTE.numRx;
chanMdl      = prmMdl.chanMdl;
chanSRate    = prmLTE.chanSRate;
corrLvl      = prmMdl.corrLevel;
switch chanMdl
   case 'flat-low-mobility',
      PathDelays = 0*(1/chanSRate);
      PathGains  = 0;
      Doppler=0;
      ChannelType =1;
   case 'flat-high-mobility',
      PathDelays = 0*(1/chanSRate);
      PathGains  = 0;
      Doppler=70;
      ChannelType =1;
   case 'frequency-selective-low-mobility',
      PathDelays = [0 10 20 30 100]*(1/chanSRate);
      PathGains  = [0 -3 -6 -8 -17.2];
      Doppler=0;
      ChannelType =1;
   case 'frequency-selective-high-mobility',
      PathDelays = [0 10 20 30 100]*(1/chanSRate);
      PathGains  = [0 -3 -6 -8 -17.2];
      Doppler=70;
      ChannelType =1;
   case 'EPA 0Hz'
      PathDelays = [0 30 70 90 110 190 410]*1e-9;
      PathGains  = [0 -1 -2 -3 -8 -17.2 -20.8];
      Doppler=0;
      ChannelType =1;
   otherwise
      ChannelType =2;
      AntConfig=char([48+numTx,'x',48+numRx]);
end
% Initialize objects
persistent chanObj;
if isempty(chanObj)
   if ChannelType ==1
```

```
    chanObj = comm.MIMOChannel('SampleRate', chanSRate, ...
      'MaximumDopplerShift', Doppler, ...
      'PathDelays', PathDelays,...
      'AveragePathGains', PathGains,...
      'RandomStream', 'mt19937ar with seed',...
      'Seed', 100,...
      'NumTransmitAntennas', numTx,...
      'TransmitCorrelationMatrix', eye(numTx),...
      'NumReceiveAntennas', numRx,...
      'ReceiveCorrelationMatrix', eye(numRx),...
      'PathGainsOutputPort', true,...
      'NormalizePathGains', true,...
      'NormalizeChannelOutputs', true);
  else
    chanObj = comm.LTEMIMOChannel('SampleRate', chanSRate, ...
      'Profile', chanMdl, ...
      'AntennaConfiguration', AntConfig, ...
      'CorrelationLevel', corrLvl,...
      'RandomStream', 'mt19937ar with seed',...
      'Seed', 100,...
      'PathGainsOutputPort', true);
  end
end
[y, yPg] = step(chanObj, in);
```

除了衰落信道之外，我们的仿真也需要添加 AWGN 信道。下一个函数为用于本书的 AWGN 信道模型。它的第一个输入为输入信号（u），第二个输入为噪声功率（noiseVar）。

Algorithm

MATLAB function

```
function y = AWGNChannel(u, noiseVar )
%% Initialization
persistent AWGN
if isempty(AWGN)
  AWGN          = comm.AWGNChannel('NoiseMethod', 'Variance', ...
  'VarianceSource', 'Input port');
end
y = step(AWGN, u, noiseVar);
end
```

5.13 OFDM 接收端

我们将 OFDM 接收端处理视为发射端的反向操作。首先对每个子载波移除

CP 并进行 FFT 恢复接受信号和参考信号。根据信道带宽的不同使用不同的 FFT 长度。通过合并缩放、重组、移除 DC 子载波，解包，接收的调制符号按发射端资源网格的顺序放置。下面的 MATLAB 函数表现了 OFDM 接收端一系列操作。函数输入为接收端输入信号（in）和包含 PDSCH 参数的结构（prmLTE）。输出为接收端复原的资源网格。

Algorithm

MATLAB function

```
function y = OFDMRx(in, prmLTE)
%#codegen
persistent hFFT;
if isempty(hFFT)
   hFFT = dsp.FFT;
end
% For a subframe of data
numDataTones = prmLTE.Nrb*prmLTE.Nrb_sc;
numSymb = prmLTE.Ndl_symb*2;
[~, numLayers] = size(in);
% N assumes 15KHz subcarrier spacing, else N = 4096
N = prmLTE.N;
cpLen0 = prmLTE.cpLen0;
cpLenR = prmLTE.cpLenR;
slotLen = (N*7 + cpLen0 + cpLenR*6);
tmp = complex(zeros(N, numSymb, numLayers));
% Remove CP - unequal lengths over a slot
for j = 1:2 % over two slots
   % First OFDM symbol
   tmp(:, (j-1)*7+1, :) = in((j-1)*slotLen+cpLen0 + (1:N), :);

   % Next 6 OFDM symbols
   for k = 1:6
      tmp(:, (j-1)*7+k+1, :) = in((j-1)*slotLen+cpLen0+k*N+k*cpLenR + (1:N), :);
   end
end
% FFT processing
x = step(hFFT, tmp);
x =  x./(N/sqrt(numDataTones));
% For a subframe of data
y = complex(zeros(numDataTones, numSymb, numLayers));
% Reorder, remove DC, Unpack data
x = [x(N/2+1:N, :, :); x(1:N/2, :, :)];
y(1:(numDataTones/2), :, :) = x(N/2-numDataTones/2+1:N/2, :, :);
y(numDataTones/2+1:numDataTones, :, :) = x(N/2+2:N/2+1+numDataTones/2, :, :);
end
```

5.14 资源元素反映射

资源元素反映射为资源网格映射的反向操作。下面的 MATLAB 函数表现了参考信号和数据如何在接收端从复原的资源网格中解压。函数有三个输入：接收资源网格（in）、子帧索引（nS），和 PDSCH 参数设定。函数输出为用户数据（data）、用户数据索引（idx_ data）、CSR 信号（csr），和可选的 DCI（pdcch）、PSS 和 SSS（pss，sss），以及 BCH（bch）信号。不同的子帧包含不同的内容，第二个输入子帧索引参数（nS）可以让函数分离正确数据。资源映射函数的算法同样在反映射函数中用于生成索引。

Algorithm

MATLAB function

```
function [data, csr, idx_data, pdcch, pss, sss, bch] = REdemapper_1Tx(in, nS, prmLTE)
%#codegen
% NcellID = 0;                          % One of possible 504 values
% numTx = 1;                            % prmLTE.numTx;
% Get input params
Nrb = prmLTE.Nrb;                       % either of {6, }
Nrb_sc = prmLTE.Nrb_sc;                 % 12 for normal mode
numContSymb    = prmLTE.contReg;  % either {1, 2, 3}
%% Specify resource grid location indices for CSR, PDCCH, PDSCH, PBCH, PSS, SSS
%% 1st: Indices for CSR pilot symbols
lenOFDM = Nrb*Nrb_sc;
idx          = 1:lenOFDM;
idx_csr0   = 1:6:lenOFDM;          % More general starting point = 1+mod(NcellID, 6);
idx_csr4   = 4:6:lenOFDM;          % More general starting point = 1+mod(3+NcellID, 6);
idx_csr    =[idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0, 11*lenOFDM+idx_csr4];
%% 2nd: Indices for PDCCH control data symbols
ContREs=numContSymb*lenOFDM;
idx_dci=1:ContREs;
idx_pdcch = ExpungeFrom(idx_dci,idx_csr0);
%% 3rd: Indices for PDSCH and PDSCH data in OFDM symbols where pilots are present
idx_data0= ExpungeFrom(idx,idx_csr0);
idx_data4 = ExpungeFrom(idx,idx_csr4);
%% Handle 3 types of subframes differently
pss=complex(zeros(72,1));
sss=complex(zeros(72,1));
bch=complex(zeros(72*4,1));
switch nS
   %% 4th: Indices for BCH, PSS, SSS are only found in specific subframes 0 and 5
   % These symbols share the same 6 center sub-carrier locations (idx_ctr)
   % and differ in OFDM symbol number.
   case 0 % Subframe 0
      % PBCH, PSS, SSS are available + CSR, PDCCH, PDSCH
```

```
        idx_6rbs = (1:72);
        idx_ctr = 0.5* lenOFDM - 36 + idx_6rbs ;
        idx_SSS  = 5* lenOFDM + idx_ctr;
        idx_PSS  = 6* lenOFDM + idx_ctr;
        idx_ctr0 = ExpungeFrom(idx_ctr,idx_csr0);
        idx_bch=[7*lenOFDM + idx_ctr0, 8*lenOFDM + idx_ctr, 9*lenOFDM + idx_ctr,
10*lenOFDM + idx_ctr];
        idx_data5  = ExpungeFrom(idx,idx_ctr);
        idx_data7 = ExpungeFrom(idx_data0,idx_ctr);
        idx_data  = [ContREs+1:4*lenOFDM,  4*lenOFDM+idx_data4, ...
          5*lenOFDM+idx_data5, 6*lenOFDM+idx_data5,  7*lenOFDM+idx_data7,
8*lenOFDM+idx_data5, ...
          9*lenOFDM+idx_data5, 10*lenOFDM+idx_data5, 11*lenOFDM+idx_data4, ...
          12*lenOFDM+1:14*lenOFDM];
        pss=in(idx_PSS).';   % Primary Synchronization Signal (PSS)
        sss=in(idx_SSS).';   % Secondary Synchronization Signal (SSS)
        bch=in(idx_bch).';   % Broadcast Channel data (BCH)

    case 10 % Subframe 5
        % PSS, SSS are available + CSR, PDCCH, PDSCH
        % Primary and Secondary synchronization signals in OFDM symbols 5 and 6
        idx_6rbs = (1:72);
        idx_ctr = 0.5* lenOFDM - 36 + idx_6rbs ;
        idx_SSS  = 5* lenOFDM + idx_ctr;
        idx_PSS  = 6* lenOFDM + idx_ctr;
        idx_data5 = ExpungeFrom(idx,idx_ctr);
        idx_data  = [ContREs+1:4*lenOFDM, 4*lenOFDM+idx_data4,
5*lenOFDM+idx_data5, 6*lenOFDM+idx_data5, ...
          7*lenOFDM+idx_data0, 8*lenOFDM+1:11*lenOFDM,  11*lenOFDM+idx_data4, ...
          12*lenOFDM+1:14*lenOFDM];
        pss=in(idx_PSS).';   % Primary Synchronization Signal (PSS)
        sss=in(idx_SSS).';   % Secondary Synchronization Signal (SSS)

    otherwise % other subframes
        % Only CSR, PDCCH, PDSCH
        idx_data = [ContREs+1:4*lenOFDM, 4*lenOFDM+idx_data4, ...
          5*lenOFDM+1:7*lenOFDM, ...
          7*lenOFDM+idx_data0, ...
          8*lenOFDM+1:11*lenOFDM, ...
          11*lenOFDM+idx_data4, ...
          12*lenOFDM+1:14*lenOFDM];
end
data=in(idx_data).';       % Physical Downlink Shared Channel (PDSCH) = user data
csr=in(idx_csr).';         % Cell-Specific Reference signal (CSR) = pilots
pdcch = in(idx_pdcch).';  % Physical Downlink Control Channel (PDCCH)
end
```

5.15 信道估计

信道估计即测量参考符号，或在 OFDM 时 - 频网格中间隙中插入的导频。使用已知参考符号，接收端可以在发送该参考符号的子载波上进行信道估计。参考符号应该在时域和频域有足够高的密度。如此，通过适当的展宽我们可以得到全部时 - 频网格的估计。

下面的 MATLAB 函数表现了单天线传输的信道估计。输入为包含 PDSCH 参数的结构（prmLTE）、接收的资源网格（Rx）、CSR（ref）和频带展宽模式（mode）。在对接收的资源网格进行整形之后，接收信号按 CSR 内相应的导频信号进行排列。我们随后用接收到的发射参考信号计算信道相应矩阵（hD）的估计。之后通过计算所有按 CSR 信号排列的资源元素的信道相应矩阵，我们得到全频带展宽的结果。基于资源网格中参考信号的子集，我们可以通过对全部资源网格求信道估计的平均或内插值得到展宽结果；这一过程在每一个子载波和每一个子帧的 OFDM 符号上进行。

Algorithm

MATLAB function

```
function hD = ChanEstimate_1Tx(prmLTE, Rx, Ref, Mode)
%#codegen
Nrb         = prmLTE.Nrb;     % Number of resource blocks
Nrb_sc      = prmLTE.Nrb_sc;              % 12 for normal mode
Ndl_symb = prmLTE.Ndl_symb;       % 7   for normal mode
% Assume same number of Tx and Rx antennas = 1
% Initialize output buffer
hD = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2));
% Estimate channel based on CSR - per antenna port
csrRx = reshape(Rx, numel(Rx)/4, 4); % Align received pilots with reference pilots
hp      = csrRx./Ref;                 % Just divide received pilot by reference pilot
% to obtain channel response at pilot locations
% Now use some form of averaging/interpolation/repeating to
% compute channel response for the whole grid
% Choose one of 3 estimation methods "average" or "interpolate" or "hybrid"
switch Mode
  case 'average'
    hD=gridResponse_averageSubframe(hp, Nrb, Nrb_sc, Ndl_symb);
  case 'interpolate'
    hD=gridResponse_interpolate(hp, Nrb, Nrb_sc, Ndl_symb);
  otherwise
    error('Choose the right mode for function ChanEstimate.');
end
end
```

典型内插算法针对含有 CSR 信号（子帧 0、5、7 和 12）OFDM 符号频域上的两个子载波。通过计算特定符号的所有子载波信道相应，我们可以内插找到整个网格的信道相应。下面的 MATLAB 函数（gridResponse_ interpolate）进行基于内插的展宽算法。

Algorithm

MATLAB function

```
function hD=gridResponse_interpolate(hp, Nrb, Nrb_sc, Ndl_symb)
% Interpolate among subcarriers in each OFDM symbol
% containing CSR (Symbols 1,5,8,12)
% The interpolation assumes NCellID = 0.
% Then interpolate between OFDM symbols
hD = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2));
N=size(hp,2);
Separation=6;
Edges=[0,5;3,2;0,5;3,2];
Symbol=[1,5,8,12];
% First: Compute channel response over all resource elements of OFDM symbols 0,4,7,11
for n=1:N
   Edge=Edges(n,:);
   y = InterpolateCsr(hp(:,n),  Separation, Edge);
   hD(:,Symbol(n))=y;
end
% Second: Interpolate between OFDM symbols {0,4} {4,7}, {7, 11}, {11, 13}
for m=[2, 3, 4, 6, 7]
   alpha=0.25*(m-1);
   beta=1-alpha;
   hD(:,m)    = beta*hD(:,1) + alpha*hD(:,  5);
   hD(:,m+7) =beta*hD(:,8) + alpha*hD(:,12);
end
```

典型的平均算法在含有 CSR 信号（子帧 0、5、7 和 12）OFDM 符号频域上的两个子载波间进行内插。首先我们从开始两个 OFDM 符号（子帧 0 和 5）上合并 CSR 信号。不同于在两个 CSR 信号间间隔六个子载波，这一过程使两个 CSR 间隔三个子载波。随后我们在频轴进行内插。最后我们对时隙或子帧上 OFDM 符号用相同的信道响应寻找全体网格的信道响应。下面的 MATLAB 函数（gridResponse_ averageSubframe）展示了这一过程。

Algorithm

MATLAB function

```
function hD=gridResponse_averageSubframe(hp, Nrb, Nrb_sc, Ndl_symb)
% Average over the two same Freq subcarriers, and then interpolate between
% them - get all estimates and then repeat over all columns (symbols).
```

```
% The interpolation assumes NCellID = 0.
% Time average two pilots over the slots, then interpolate (F)
% between the 4 averaged values, repeat for all symbols in subframe
h1_a1 = mean([hp(:, 1), hp(:, 3)],2);
h1_a2 = mean([hp(:, 2), hp(:, 4)],2);
h1_a_mat = [h1_a1 h1_a2].';
h1_a = h1_a_mat(:);
h1_all = complex(zeros(length(h1_a)*3,1));
for i = 1:length(h1_a)-1
   delta = (h1_a(i+1)-h1_a(i))/3;
   h1_all((i-1)*3+1) = h1_a(i);
   h1_all((i-1)*3+2) = h1_a(i)+delta;
   h1_all((i-1)*3+3) = h1_a(i)+2*delta;
end
% fill the last three - use the last delta
h1_all(end-2) = h1_a(end);
h1_all(end-1) = h1_a(end)+delta;
h1_all(end) = h1_a(end)+2*delta;
% Compute the channel response over the whole grid by repeating
hD = h1_all(1:Nrb*Nrb_sc, ones(1, Ndl_symb*2));
end
```

5.16 均衡器增益计算

一个频域均衡器对所有每个子载波接收的资源元素计算增益。频域均衡有不同的算法。最简单的为 ZF 算法，它通过发射资源元素和每个子载波信道估计的比值得到增益。更加复杂一些的算法为 MMSE 估计，它基于更复杂的信道时/频特性理论计算包含非相关性信道噪声的比值得到增益。当得到均衡器增益之后，通过接收资源元素和均衡器增益的乘积即可得到资源元素的最优估计。下面的 MATLAB 函数中，用户可通过选择均衡模式参数选择使用 ZF 或 MMSE 均衡器。

Algorithm

MATLAB function

```
function [out, Eq] = Equalizer(in, hD, nVar, EqMode)
%#codegen
switch EqMode
   case 1,
     Eq = ( conj(hD))./((conj(hD).*hD));          % Zero forcing
   case 2,
      Eq = ( conj(hD))./((conj(hD).*hD)+nVar);  % MMSE
   otherwise,
      error('Two equalization mode vaible: Zero forcing or MMSE');
end
out=in.*Eq;
```

5.17 信道可视化

可视化各种信号可以帮助我们验证 OFDM 是否正确传输。在 OFDM 中，每个调制符号由 OFDM 符号（时域）的子载波（频域）传输。这使我们可以在通过信道前后直接观察传输的衰落效应。在下面的 MATLAB 函数中，我们使用 DSP 系统工具箱中的频谱分析系统对象有效的获取发射端和接收端的数据频谱。该函数输入为变量 txSig 和 rxSig，它们表示通过信道模型前后的 OFDM 调制信号。另一个变量 yRec 表示均衡之后的用户数据。通过频谱分析器可视化这三个变量，我们可以观察信道模型对传输信号的影响以及接收端信道估计和均衡对复原发射信号的最优估计的影响。我们还会使用通信系统工具箱中星座图系统对象观察调制信号进行均衡前后衰落信道的对其的影响。

Algorithm

MATLAB function

```
function zVisualize(prmLTE, txSig, rxSig, yRec, dataRx, csr, nS)
% Constellation Scopes & Spectral Analyzers
persistent hScope1 hScope2 hSpecAnalyzer
if isempty(hSpecAnalyzer)
   % Constellation Diagrams
   hScope1 = comm.ConstellationDiagram('SymbolsToDisplay',...
      prmLTE.numDataResources, 'ShowReferenceConstellation', false,...
      'YLimits', [-2 2], 'XLimits', [-2 2], 'Position', ...
      figposition([5 60 20 25]), 'Name', 'Before Equalizer');
   hScope2 = comm.ConstellationDiagram('SymbolsToDisplay',...
      prmLTE.numDataResources, 'ShowReferenceConstellation', false,...
      'YLimits', [-2 2], 'XLimits', [-2 2], 'Position', ...
      figposition([6 61 20 25]), 'Name', 'After Equalizer');
   % Spectrum Scope
   hSpecAnalyzer = dsp.SpectrumAnalyzer('SampleRate', prmLTE.chanSRate, ...
      'SpectrumType', 'Power density', 'PowerUnits', 'dBW', ...
      'RBWSource', 'Property', 'RBW', 15000,...
      'FrequencySpan', 'Span and center frequency',...
      'Span', prmLTE.BW, 'CenterFrequency', 0,...
      'FFTLengthSource', 'Property', 'FFTLength', prmLTE.N,...
      'Title', 'Transmitted & Received Signal Spectrum', 'YLimits', [-110 -60],...
      'YLabel', 'PSD');
end
% Update Spectrum scope
% Received signal after equalization
yRecGrid = REmapper_1Tx(yRec, csr, nS, prmLTE);
```

```
yRecGridSig = lteOFDMTx(yRecGrid, prmLTE);
% Take certain symbols off a subframe only
step(hSpecAnalyzer, ...
   [SymbSpec(txSig, prmLTE), SymbSpec(rxSig, prmLTE),
SymbSpec(yRecGridSig, prmLTE)]);
% Update Constellation Scope
if (nS~=0 && nS~=10)
   step(hScope1, dataRx(:, 1));
   step(hScope2, yRec(:, 1));
end
end
% Helper function
function y = SymbSpec(in, prmLTE)
N = prmLTE.N;
cpLenR = prmLTE.cpLen0;
y = complex(zeros(N+cpLenR, 1));
% Use the first Tx/Rx antenna of the input for the display
y(:,1) = in(end-(N+cpLenR)+1:end, 1);
end
```

5.18 下行链路传输模式 1

在本节中，我们会将前两张中构建的函数们组合构成 LTE 下行链路模式 1 的模型。模式 1 基于单天线传输。我们会针对这个模式构建两种模型。

1）SISO 模型：收发端皆为单天线配置；

2）SIMO 模型：使用单发射天线和多接收天线，以优化接收分集。

在本书中，我们的每个 PHY 信号处理模型都包括发射端、信道模型，和接收端。在本章中，发射端处理包括下行链路公共信道（DLSCH）和 PDSCH。信道模型为衰落信道和 AWGN 信道的组合。接收端反向 DLSCH 和 PDSCH 处理。

仿真单位为子帧。因用户数据由每个子帧搭载和处理，我们监控子帧索引以保证不同子帧索引可被正确处理。子帧索引值递增到所有帧处理完成后归零。该处理在整个仿真的帧处理过程中重复进行。仿真 LTE mode1 模型的程序由两部分构成。

1）MATLAB 函数：包含所有发射端，信道模型和接收端的数据单子帧操作。

2）MATLAB 测试脚本：初始化所有 DLSCH、PDSCH，和信道模型的参数，然后循环处理所有子帧并计算字节误码率（BER）。当满足最大误码数或最大字节数条件时停止仿真。

5.18.1 SISO 模型

下面的 MATLAB 函数为 SISO 收发端（发射端、信道模型，和接收端）处理

模型。信号在发射端的 DLSCH 和 PDSCH 处理过程如下：

1）生成单子帧载荷数据（传输块）。

2）DLSCH 处理，包括：传输块添加 CRC，码组分割和 CRC 添加，1/3 码率 Turbo 编码，码率匹配，和码组连接构成 PDSCH 输入码字。

3）PDSCH 处理，包括：绕码，对绕码后的字节进行调制，映射复调制符号到资源元素在单天线端口形成资源网格，生成 OFDM 信号。

信道模型为衰落信道和 AWGN 信道的组合。在接收端，反向 PDSCH 操作，包括：OFDM 信号接收端生成资源网格，资源元素从用户数据反映射 CSR，并按 CSR 进行信道估计和频域均衡，软判决解调和去绕码。

最后，接收端反向 DLSCH，包括：码组分割，码率去匹配，和 CRC 早期终止机制 Turbo 译码。接收端输出变量 data_out 和发射端输入传输块变量 dataIn 为函数最先生命的两个输出。除了这两个变量，几个其他输出变量用来提升测试系统任务的性能。我们将会在下面进行一些定量和定性的性能分析。

Algorithm

MATLAB function

```
function [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr]...
    = commlteSISO_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl)
%% TX
% Generate payload
dataIn = genPayload(nS, prmLTEDLSCH.TBLenVec);
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
[data, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH, prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
% Generate Cell-Specific Reference (CSR) signals
csr = CSRgenerator(nS, prmLTEPDSCH.numTx);
% Resource grid filling
E=8*prmLTEPDSCH.Nrb;
csr_ref=reshape(csr(1:E),2*prmLTEPDSCH.Nrb,4);
txGrid = REmapper_1Tx(modOut, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
%% Channel
% SISO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
idealhD = lteIdChEst(prmLTEPDSCH, prmMdl, chPathG, nS);
```

```
% Add AWG noise
nVar = 10.^(0.1.*(-snrdB));
rxSig =  AWGNChannel(rxFade, nVar);
%% RX
% OFDM Rx
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = REdemapper_1Tx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
   chEst = ChanEstimate_1Tx(prmLTEPDSCH, csrRx,  csr_ref, 'interpolate');
   hD=chEst(idx_data).';
else
   hD = idealhD;
end
% Frequency-domain equalizer
yRec = Equalizer(dataRx, hD, nVar, prmLTEPDSCH.Eqmode);
% Demodulate
demodOut = DemodulatorSoft(yRec, prmLTEPDSCH.modType, nVar);
% Descramble both received codewords
rxCW =  lteDescramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes - CB segmentation, turbo decoding, rate dematching
[decTbData1, ~,~] = lteTbChannelDecoding(nS, rxCW, Kplus1, C1,  prmLTEDLSCH,
prmLTEPDSCH);
% Transport block CRC detection
[dataOut, ~] = CRCdetector(decTbData1);
end
```

5.18.1.1 收发端模型结构

下面的 MATLAB 脚本调用 SISO 函数。首先，它进行初始化（commlteSISO_initialize），为三个 MATLAB 结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）设置所有与 DLSCH、PDSCH 和信道模型有关的参数，随后它启动子帧处理循环。在 While 循环之前，初始化子帧索引（nS）以保证当帧数据（10ms）处理结束之后索引归零。这个循环内也包含仿真终止条件（最大处理字节数和最大误码数）。脚本也比较输入和输出以计算 BER，并调用可视化函数直观评估均衡前后的信道响应和调制星座图。

Algorithm

MATLAB script: commlteSISO

```
% Script for SISO LTE (mode 1)
% Single codeword transmission only,
clear all
clear functions
disp('Simulating the LTE Mode 1: Single Tx and Rx antrenna');
%% Set simulation parametrs & initialize parameter structures
```

```
commlteSISO_params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteSISO_initialize( chanBW,
contReg,  modType, Eqmode,...
   cRate,maxIter, fullDecode, chanMdl, corrLvl, chEstOn, maxNumErrs, maxNumBits);
clear chanBW contReg numTx numRx modType Eqmode cRate maxIter fullDecode
chanMdl corrLvl chEstOn maxNumErrs maxNumBits;
%%
hPBer = comm.ErrorRate;
iter=numel(prmMdl.snrdBs);
snrdB=prmMdl.snrdBs(iter);
maxNumErrs=prmMdl.maxNumErrs(iter);
maxNumBits=prmMdl.maxNumBits(iter);
%% Simulation loop
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
  [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
     commlteSISO_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
   % Calculate  bit errors
   Measures = step(hPBer, dataIn, dataOut);
    % Visualize constellations and spectrum
   if visualsOn, zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);end;
   % Update subframe number
   nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
end
disp(Measures);
```

以下的初始化函数设置了关键仿真参数。因程序针对 SISO 模型，收发天线数设定为 1。PDSCH 设定允许用户从 6 个支持选项中选择指定带宽（chanBW），OFDM 占据控制区符号数（contReg），三种调制类型中选一（modType），以及均衡算法。DLSCH 参数设置选取输入参数为码率（cRate），Turbo 译码器最大循环数（maxIter），和是否在全 Turbo 译码过程中使用早期终止机制（fullDecode）。最后函数设定控制信道模型的参数，包括信道模式类型（chanMdl），相邻天线端口的相关性程度（corrLvl），和是否进行信道评估或理想信道评估（chEstOn）。

Algorithm

MATLAB function

```
function [prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteSISO_initialize(chanBW,
contReg, modType, Eqmode,...
                          cRate,maxIter, fullDecode, chanMdl, corrLvl, chEstOn,
maxNumErrs, maxNumBits)
% Create the parameter structures
% PDSCH and DLSCH
```

```
prmLTEPDSCH = prmsPDSCH(chanBW, contReg, modType);
prmLTEPDSCH.Eqmode=Eqmode;
prmLTEPDSCH.modType=modType;
prmLTEDLSCH = prmsDLSCH(cRate,maxIter, fullDecode, prmLTEPDSCH);
% Channel parameters
prmMdl.chanMdl = chanMdl;
prmMdl.corrLevel = corrLvl;
prmMdl.chEstOn = chEstOn;
switch modType
   case 1
      snrdBs=[0:4:8, 9:12];
   case 2
      snrdBs=[0:4:12, 13:16];
   otherwise
      snrdBs=0:4:24;
end
prmMdl.snrdBs=snrdBs;
prmMdl.maxNumBits=maxNumBits*ones(size(snrdBs));
prmMdl.maxNumErrs=maxNumErrs*ones(size(snrdBs));
```

5.18.1.2 收发端性能验证

通过执行 SISO 收发模型（commlteSISO）的 MATLAB 测试脚本，我们可以观察各种信号以评估系统性能。为了执行模型脚本，我们首先需要设定与各个模型组件相关的参数。下面这个脚本（commlteSISO_params）设定了相关参数，包括设定调制器调制类型为 16QAM。

Algorithm

MATLAB script

```
% PDSCH
numTx         = 1;   % Number of transmit antennas
numRx         = 1;   % Number of receive antennas
chanBW        = 4;   % Index to chanel bandwidth used [1,....6]
contReg       = 1;   % No. of OFDM symbols dedictaed to control information [1,...,3]
modType       = 2;   % Modulation type [1, 2, 3] for ['QPSK','16QAM','64QAM']
% DLSCH
cRate         = 1/3; % Rate matching target coding rate
maxIter       = 6;   % Maximum number of turbo decoding terations
fullDecode    = 0;   % Whether "full" or "early stopping" turbo decoding is performed
% Channel model
chanMdl       = 'frequency-selective-high-mobility';
corrLvl       = 'Low';
% Simulation parametrs
Eqmode        = 2;   % Type of equalizer used [1,2] for ['ZF', 'MMSE']
chEstOn       = 1;   % Whether channel estimation is done or ideal channel model used
maxNumErrs = 5e7; % Maximum number of errors found before simulation stops
maxNumBits = 5e7; % Maximum number of bits processed before simulation stops
visualsOn     = 1;   % Whether to visualize channel response and constellations
```

例如，为了考量均衡的影响，我们用星座图观察接收端均衡前后复原用户数据的情况。作为 commlteSISO_ step 函数的输出，我们可以观察到变量 dataRx 和 yRec。图 5.11 所示的星座图展示了均衡器可以补偿衰落信道（左图）并使均衡后的星座图更接近 16QAM 调制的星座图（右图）。

为了衡量 OFDM 接收端多径衰落效应的影响，我们观察发射信号和接收信号均衡前后的功率谱密度。我们可以观察 MATLAB 变量（txSig，rxSig 和 yRec）。图 5.12 所示为发射信号、接收信号均衡前与接收信号均衡后的功率谱。结果显示发射信号频谱归一化后，接收信号的频谱反映了信道多径衰落的情况。当均衡之后，衰落效应最大限度的减小了，其功率谱显示频率平坦接近发射信号频谱。

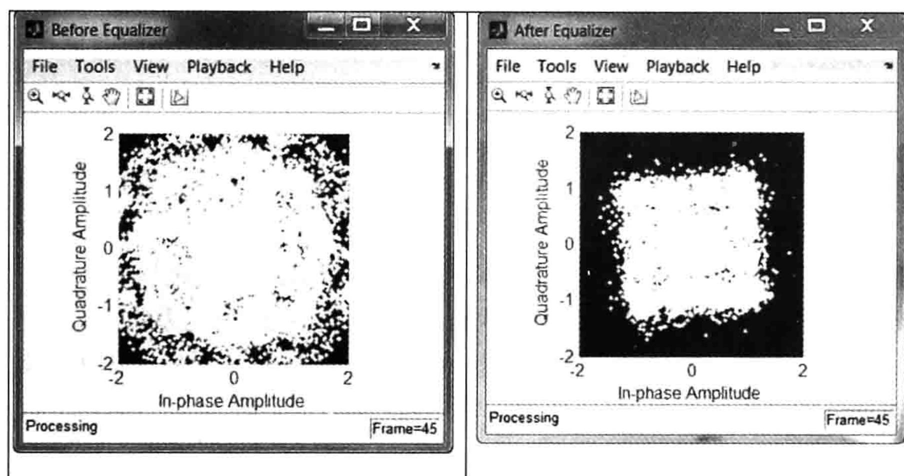


图 5.11 LTE SISO 模型：用户数据均衡前后的星座图

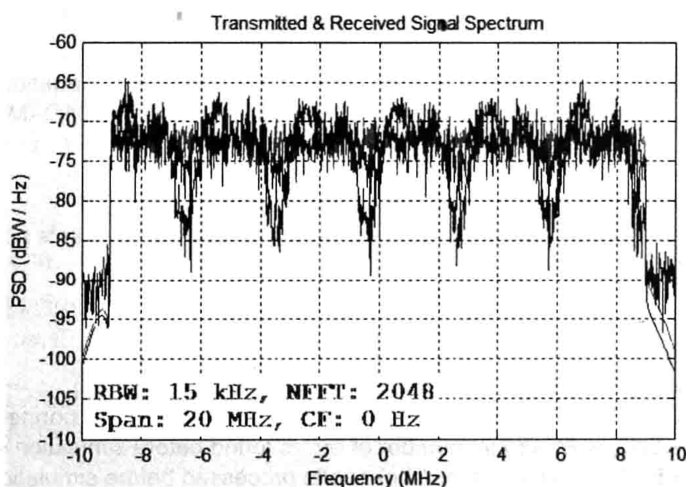


图 5.12 LTE SISO 模型：接收信号均衡前后与发射信号的功率谱

5.18.1.3 BER 测量

为了验证收发端的 BRE 性能，我们建立一个测试平台命名为 commlteSISO_test_timing_ber.m。它首先初始化 LTE 系统参数，然后循环遍历信噪比（SNR）值并调用函数 commlteSISO_fcn 计算相应的 BER 值。脚本同时使用 MATLAB tic 和 toc 函数测量完成循环计算的时间。

Algorithm

MATLAB script: commlteSISO_test_timing_ber

```
% Script for SISO LTE (mode 1)
%
% Single codeword transmission only,
%
clear all
clear functions
disp('Simulating the LTE Mode 1: Single Tx and Rx antrenna');
%% Create the parameter structures
commlteSISO_params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteSISO_initialize( chanBW,
contReg, modType, Eqmode,...
   cRate,maxIter, fullDecode, chanMdl, corrLvl, chEstOn, maxNumErrs, maxNumBits);
clear  chanBW contReg numTx numRx modType Eqmode cRate maxIter fullDecode
chanMdl corrLvl chEstOn maxNumErrs maxNumBits;
%%
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
MaxIter=numel(prmMdl.snrdBs);
ber_vector=zeros(1,MaxIter);
tic;
for n=1:MaxIter
   fprintf(1,'Iteration %2d out of %2d\n', n, MaxIter);
   [ber, ~] = commlteSISO_fcn(n, prmLTEPDSCH, prmLTEDLSCH, prmMdl);
   ber_vector(n)=ber;
end;
toc;
```

当 MATLAB 脚本执行时，命令窗口内会出现包括收发端参数（调制类型、编码率、带宽、天线配置，最大数据速率），循环正在执行，和总运行时间的信息。

图 5.13 显示了收发端 BER 和 SNR 的关系。在本例中，我们对每个 SNR 值运行 8 个循环，每 8 个循环处理 5 千万比特。接收端使用 16QAM 调制方案，1/3 码率，10MHz 系统带宽，和 SISO（1x1）天线配置。通过设置这些参数得到最大数据速率为 9.91Mbit/s，如 zReport_data_rate.m 函数得到的结果。

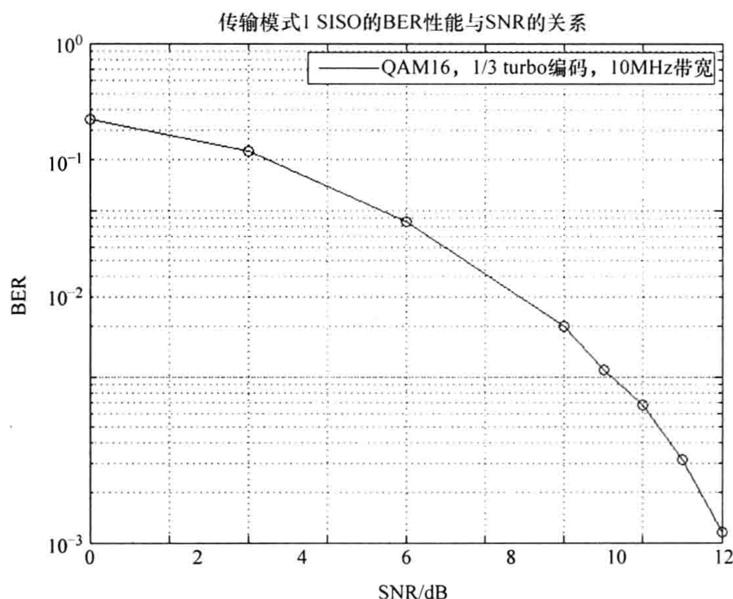


图 5.13 BER 结果：SISO 模式

5.18.2 SIMO 模型

SIMO 模式可以看作是 SISO 模式的一般情况。LTE 传输模式 1 一般使用 SIMO 模式。在这个模式中，信号处理过程和 SISO 类似，除了使用多路（在我们的函数中为 2 和 4）接收天线。在接收端使用多天线可以发挥接收分集的优势。最大比合并（MRC）接收分集可使系统的 BER 特性好与 SISO。接收分集建模并不需要改变发射端，不过在信道建模和接收端有较大改变。这些改变与多信道处理有关。

发射端操作之后，衰落信道从单发射天线处理采样。根据接收天线数量的不同，信道模型分别影响每个链路（收发对）。衰落信道的输出为一个多信道矩阵，它的行数等于发射采样数而列数等于接收天线数。相似的，AWGN 信道作用于衰落信道多信道输出矩阵生成同样大小携带白噪声的输出矩阵。

多信道接收信号作为接收端的输入，首先接收端必须进行重复扫描所有不同信道的操作（即不同接收天线）。这一过程涉及了 OFDM 接收器、资源元素反映射器，和信道估计器和均衡器。

在每个接收端估计数据资源元素结合了新均衡器用于产生发射信号的最好估计。均衡器在每个天线使用 ZF 和 MMSE 方法，其结果按 MRC 合并。这一方法中，每个接收天线根据功率测量发挥了更大的作用。下面的 MATLAB 函数为 SIMO 收发端模型。

Algorithm

MATLAB function

```
function [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr]...
   = commlteSIMO_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl)
%% TX
% Generate payload
dataIn = genPayload(nS, prmLTEDLSCH.TBLenVec);
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn);
% Channel coding includes – CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation – per codeword
[data, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH, prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
% Generate Cell-Specific Reference (CSR) signals
csr = CSRgenerator(nS, prmLTEPDSCH.numTx);
% Resource grid filling
E=8*prmLTEPDSCH.Nrb;
csr_ref=reshape(csr(1:E),2*prmLTEPDSCH.Nrb,4);
txGrid = Remapper_1Tx(modOut, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
%% Channel
% SISO Fading channel
numRx=prmLTEPDSCH.numRx;
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
idealhD = lteIdChEst(prmLTEPDSCH,  prmMdl, chPathG, nS);
% Add AWG noise
nVar = 10.^(0.1.*(-snrdB));
rxSig =  AWGNChannel(rxFade, nVar);
%% RX
% OFDM Rx
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = Redemapper_1Tx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
   chEst = ChanEstimate_1Tx(prmLTEPDSCH, csrRx,  csr_ref, 'interpolate');
   hD=complex(zeros(numel(idx_data),numRx));
   for n=1:numRx
      tmp=chEst(:,:,n);
      hD(:,n)=tmp(idx_data).';
   end
else
```

```
    hD = idealhD;
end
% Frequency-domain equalizer
% Based on Maximum-Combining Ratio (MCR)
yRec = Equalizer_simo( dataRx, hD, nVar, prmLTEPDSCH.Eqmode);
% Demodulate
demodOut = DemodulatorSoft(yRec, prmLTEPDSCH.modType, nVar);
% Descramble both received codewords
rxCW =  lteDescramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes - CB segmentation, turbo decoding, rate dematching
[decTbData1, ~,~] = lteTbChannelDecoding(nS, rxCW, Kplus1, C1,  prmLTEDLSCH,
prmLTEPDSCH);
% Transport block CRC detection
[dataOut, ~] = CRCdetector(decTbData1);
end
```

5.18.2.1 改进函数

SIMO 模型函数需要改进下面的三个函数。

Redemapper_1Tx 现在支值了多信道处理，它通过在 for 循环内扫描接收天线分别解压数据、CRC，和其他信号。

Algorithm

MATLAB function

```
function [data, csr, idx_data, pdcch, pss, sss, bch] = Redemapper_1Tx(in, nS, prmLTE)
%#codegen
% NcellID = 0;                          % One of possible 504 values
% numTx = 1;                            % prmLTE.numTx;
% Get input params
numRx=prmLTE.numRx;                     % number of receive antennas
Nrb = prmLTE.Nrb;                       % either of {6,...,1}
Nrb_sc = prmLTE.Nrb_sc;                 % 12 for normal mode
numContSymb    = prmLTE.contReg;  % either {1, 2, 3}
Npss= prmLTE.numPSSRE;
Nsss=prmLTE.numSSSRE;
Nbch=prmLTE.numBCHRE;
Ncsr=prmLTE.numCSRResources;
Ndci=prmLTE.numContRE;
%% Specify resource grid location indices for CSR, PDCCH, PDSCH, PBCH, PSS, SSS
%% 1st: Indices for CSR pilot symbols
lenOFDM = Nrb*Nrb_sc;
idx          = 1:lenOFDM;
idx_csr0   = 1:6:lenOFDM;           % More general starting point = 1+mod(NcellID, 6);
idx_csr4   = 4:6:lenOFDM;           % More general starting point = 1+mod(3+NcellID, 6);
idx_csr   =[idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0, 11*lenOFDM+idx_csr4];
%% 2nd: Indices for PDCCH control data symbols
ContREs=numContSymb*lenOFDM;
```

```
idx_dci=1:ContREs;
idx_pdcch = ExpungeFrom(idx_dci,idx_csr0);
%% 3rd: Indices for PDSCH and PDSCH data in OFDM symbols where pilots are present
idx_data0= ExpungeFrom(idx,idx_csr0);
idx_data4 = ExpungeFrom(idx,idx_csr4);
switch nS
   %% 4th: Indices for BCH, PSS, SSS are only found in specific subframes 0 and 5
   % These symbols share the same 6 center sub-carrier locations (idx_ctr)
   % and differ in OFDM symbol number.
   Case 0 % Subframe 0
      % PBCH, PSS, SSS are available + CSR, PDCCH, PDSCH
      idx_6rbs = (1:72);
      idx_ctr = 0.5* lenOFDM - 36 + idx_6rbs ;
      idx_SSS  = 5* lenOFDM + idx_ctr;
      idx_PSS  = 6* lenOFDM + idx_ctr;
      idx_ctr0 = ExpungeFrom(idx_ctr,idx_csr0);
      idx_bch=[7*lenOFDM + idx_ctr0, 8*lenOFDM + idx_ctr, 9*lenOFDM + idx_ctr,
10*lenOFDM + idx_ctr];
      idx_data5  = ExpungeFrom(idx,idx_ctr);
      idx_data7 = ExpungeFrom(idx_data0,idx_ctr);
      idx_data  = [ContREs+1:4*lenOFDM,  4*lenOFDM+idx_data4, ...
         5*lenOFDM+idx_data5, 6*lenOFDM+idx_data5, 7*lenOFDM+idx_data7,
8*lenOFDM+idx_data5, ...
         9*lenOFDM+idx_data5, 10*lenOFDM+idx_data5, 11*lenOFDM+idx_data4, ...
         12*lenOFDM+1:14*lenOFDM];
   case 10 % Subframe 5
      % PSS, SSS are available + CSR, PDCCH, PDSCH
      % Primary and Secondary synchronization signals in OFDM symbols 5 and 6
      idx_6rbs = (1:72);
      idx_ctr = 0.5* lenOFDM - 36 + idx_6rbs ;
      idx_SSS  = 5* lenOFDM + idx_ctr;
      idx_PSS  = 6* lenOFDM + idx_ctr;
      idx_data5 = ExpungeFrom(idx,idx_ctr);
      idx_data  = [ContREs+1:4*lenOFDM, 4*lenOFDM+idx_data4, 5*lenOFDM+idx_data5,
6*lenOFDM+idx_data5, ...
         7*lenOFDM+idx_data0, 8*lenOFDM+1:11*lenOFDM,  11*lenOFDM+idx_data4, ...
         12*lenOFDM+1:14*lenOFDM];
   otherwise % other subframes
      % Only CSR, PDCCH, PDSCH
      idx_data = [ContREs+1:4*lenOFDM, 4*lenOFDM+idx_data4, ...
         5*lenOFDM+1:7*lenOFDM, ...
         7*lenOFDM+idx_data0, ...
         8*lenOFDM+1:11*lenOFDM, ...
         11*lenOFDM+idx_data4, ...
         12*lenOFDM+1:14*lenOFDM];
end
%% Handle 3 types of subframes differently
```

```
pss=complex(zeros(Npss,numRx));
sss=complex(zeros(Nsss,numRx));
bch=complex(zeros(Nbch,numRx));
data=complex(zeros(numel(idx_data),numRx));
csr=complex(zeros(Ncsr,numRx));
pdcch = complex(zeros(Ndci,numRx));
for n=1:numRx
   tmp=in(:,:,n);
   data(:,n)=tmp(idx_data.');       % Physical Downlink Shared Channel (PDSCH) = user data
   csr(:,n)=tmp(idx_csr.');          % Cell-Specific Reference signal (CSR) = pilots
   pdcch(:,n) = tmp(idx_pdcch.');  % Physical Downlink Control Channel (PDCCH)
   if nS==0
      pss(:,n)=tmp(idx_PSS.');    % Primary Synchronization Signal (PSS)
      sss(:,n)=tmp(idx_SSS.');     % Secondary Synchronization Signal (SSS)
      bch(:,n)=tmp(idx_bch.');    % Broadcast Channel data (BCH)
   elseif nS==10
      pss(:,n)=tmp(idx_PSS.');     % Primary Synchronization Signal (PSS)
      sss(:,n)=tmp(idx_SSS.');      % Secondary Synchronization Signal (SSS)
   end
end
```

函数 ChanEstimate_1Tx 支持多信道处理，它通过循环多天线处理 CSR 生成资源网格。

Algorithm

MATLAB function

```
function hD = ChanEstimate_1Tx(prmLTE, Rx, Ref, Mode)
%#codegen
Nrb          = prmLTE.Nrb;    % Number of resource blocks
Nrb_sc       = prmLTE.Nrb_sc;                 % 12 for normal mode
Ndl_symb = prmLTE.Ndl_symb;         % 7    for normal mode
numRx = prmLTE.numRx;
% Assume same number of Tx and Rx antennas = 1
% Initialize output buffer
hD = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2,numRx));
% Estimate channel based on CSR - per antenna port
csrRx = reshape(Rx, numel(Rx)/(4*numRx), 4, numRx); % Align received pilots with refer-
ence pilots
for n=1:numRx
   hp= csrRx(:,:,n)./Ref;                 % Just divide received pilot by reference pilot
   % to obtain channel response at pilot locations
   % Now use some form of averaging/interpolation/repeating to
   % compute channel response for the whole grid
   % Choose one of 3 estimation methods "average" or "interpolate" or "hybrid"
   switch Mode
      case 'average'
         tmp=gridResponse_averageSubframe(hp, Nrb, Nrb_sc, Ndl_symb);
```

```
        case 'interpolate'
            tmp=gridResponse_interpolate(hp, Nrb, Nrb_sc, Ndl_symb);
        otherwise
            error('Choose the right mode for function ChanEstimate.');
    end
    hD(:,:,n)=tmp;
end
```

与 SISO 模式的频域均衡器不同，SIMO 模式的均衡器需要统合所有天线的贡献。新的均衡器（Equalizer_simo）使用 MRC 方法在接收端生成资源元素的最好估计[4]。

Algorithm

MATLAB function

```
function [y, num, denum] = Equalizer_simo(in, hD, nVar, prmLTE)
%#codegen
EqMode=prmLTE.Eqmode;
numTx=prmLTE.numTx;
numRx=size(hD,2);
if (numTx>1), error('Equalizer_simo: edicated to single transmit antenna case.');end
if numRx==1
  switch EqMode
    case 1, % Zero forcing
      num = conj(hD);
      denum=conj(hD).*hD;
    case 2, % MMSE
      num = conj(hD);
      denum=conj(hD).*hD+nVar;
  end
else
  num = conj(hD);
  denum=conj(hD).*hD;
end
y = sum(in .*num,2)./sum(denum,2);
```

5.18.2.2 验证收发端性能

为了考察接收分集带来的性能改进，我们执行 SIMO 收发模型的 MATLAB 脚本（commlteSIMO）。首先，我们设定与各个组件相关的参数（commlteSIMO_params）。这一脚本基本与 SISO 模型里的脚本相同，我们只将接收参数由 1 改为 4。

Algorithm

MATLAB script

```
% PDSCH
numTx        = 1;   % Number of transmit antennas
numRx        = 4;   % Number of receive antennas
chanBW       = 4;   % Index to chanel bandwidth used [1,....6]
contReg      = 1;   % No. of OFDM symbols edicated to control information [1,...,3]
modType      = 2;   % Modulation type [1, 2, 3] for ['QPSK','16QAM','64QAM']
% DLSCH
cRate        = 1/3; % Rate matching target coding rate
maxIter      = 6;   % Maximum number of turbo decoding terations
fullDecode   = 0;   % Whether "full" or "early stopping" turbo decoding is performed
% Channel model
chanMdl      = 'frequency-selective-high-mobility';
corrLvl      = 'Low';
% Simulation parametrs
Eqmode       = 2;   % Type of equalizer used [1,2] for ['ZF', 'MMSE']
chEstOn      = 1;   % Whether channel estimation is done or ideal channel model used
maxNumErrs = 5e7;  % Maximum number of errors found before simulation stops
maxNumBits = 5e7;  % Maximum number of bits processed before simulation stops
visualsOn    = 1;   % Whether to visualize channel response and constellations
```

图 5.14 的星座图显示 SIMO OFDM 收发端如何补偿多径衰落效应，并旋转了劣化的星座图（均衡之前）得到了从星座图看起来可正确解调的信号（均衡之后）。图 5.15 所示为发射信号和接收信号在均衡前后的功率谱。结果下式当发射信号功率谱归一化后，接收信号功率谱反映了多径衰落响应的影响。当均衡之后，接收信号功率谱平坦很多，接近发射信号功率谱。

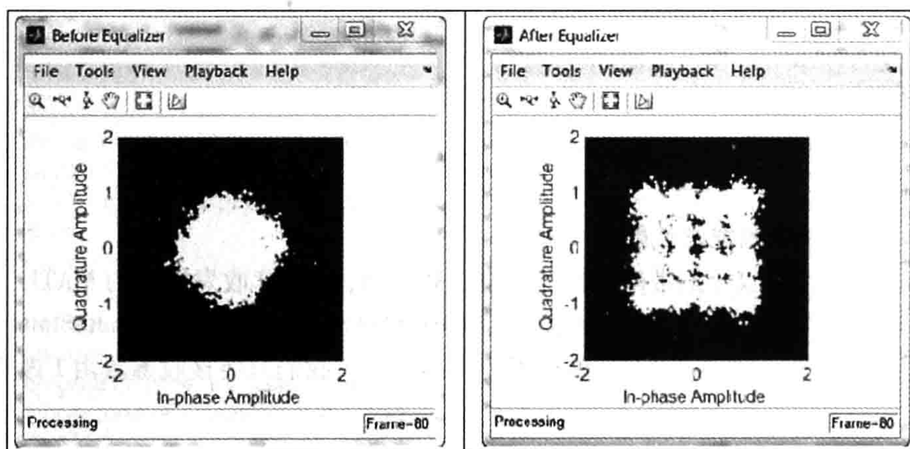


图 5.14 LTE SIMO 模型：用户数据均衡前后的星座图

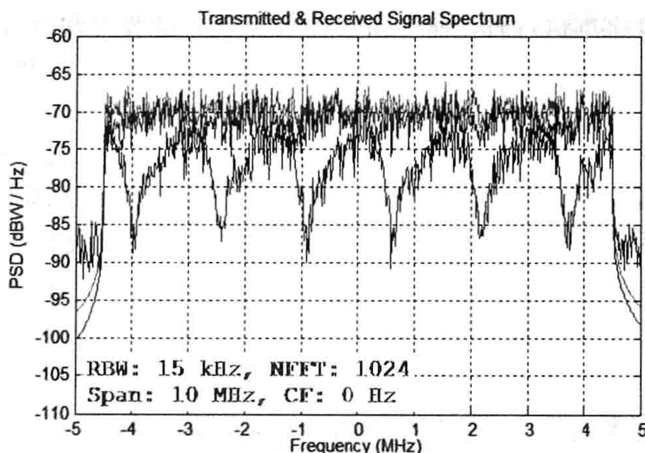


图 5.15 LTE SIMO 模型：发射信号谱密度和接收信号均衡前后的谱密度

5.18.2.3 BER 测量

为了收发端验证 BER 性能，我们创建测试脚本 commlteSIMO_test_timing_ber.m。它首先初始化 LTE 系统参数，随后遍历 SNR 值并调用 commlteSIMO_fcn 函数计算相应的 BER 值。

Algorithm

MATLAB script: *commlteSIMO_test_timing_ber*

```
% Script for SIMO LTE (mode 1)
%
% Single codeword transmission only
%
clear all
clear functions
disp('Simulating the LTE Mode 1: Single Tx and multiple Rx antrennas');
%% Set simulation parametrs & initialize parameter structures
commlteSIMO_params_ber;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteSIMO_initialize( chanBW,
contReg, modType, ...
   Eqmode, numTx, numRx, cRate,maxIter, fullDecode, chanMdl, corrLvl, ...
   chEstOn, maxNumErrs, maxNumBits);
clear chanBW contReg numTx numRx modType Eqmode cRate maxIter fullDecode
chanMdl corrLvl chEstOn maxNumErrs maxNumBits;
%%
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
MaxIter=numel(prmMdl.snrdBs);
ber_vector=zeros(1,MaxIter);
tic;
for n=1:MaxIter
   fprintf(1,'Iteration %2d out of %2d\n', n, MaxIter);
   [ber, ~] = commlteSIMO_fcn(n, prmLTEPDSCH, prmLTEDLSCH, prmMdl);
   ber_vector(n)=ber;
end;
toc;
semilogy(prmMdl.snrdBs, ber_vector);
title('BER - commlteSISO');xlabel('SNR (dB)');ylabel('ber');grid;
```

当 MATLAB 脚本执行时，命令窗口内会出现包括收发端参数（调制类型、编码率、带宽、天线配置，最大数据速率），循环正在执行，和总运行时间的信息。

图 5.16 所示为 BER 随 SNR 的变化关系。在本例中，我们对每个 SNR 值运行 8 个循环，每 8 个循环处理 5 千万个比特。接收端使用 16QAM 调制方案，1/3 码率，10MHz 系统带宽，和 SIMO（1×4）天线配置。通过设置这些参数得到最大数据速率为 9.91Mbit/s，如 zReport_data_rate.m 函数得到的结果。不采取任何加速方法运行全部八个循环需要耗时 4025s。

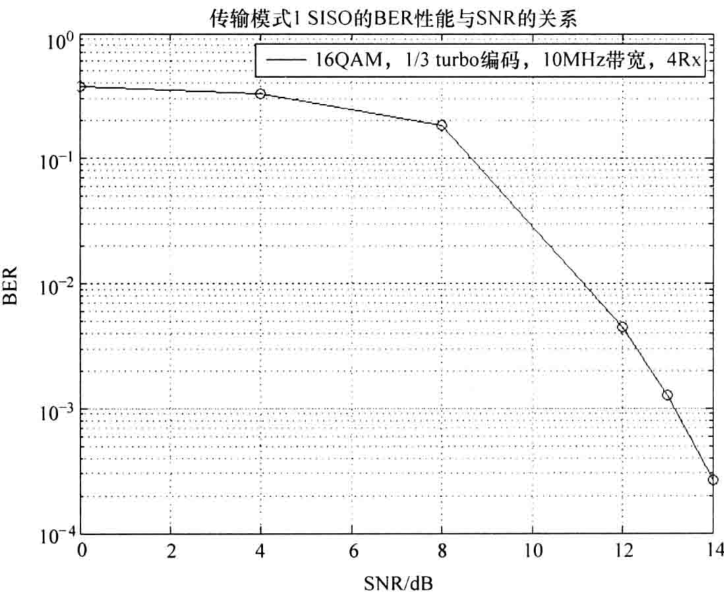


图 5.16 BER 结果：SIMO 模式

5.19 本章小结

在本章中我们研究了 LTE 标准的多载波传输方案。我们关注开发基于 OFDM 传输的下行链路收发端 MATLAB 模型。首先我们考察了各种现实移动通信信道并介绍多径衰落信道模型。随后我们引入 OFDM 传输方案的功能模块，寻找减小多径衰落的方法。

我们随后回顾了发射端的组件，包括：

1）作为资源网格的导引，介绍数据时 - 频分布；

2）介绍资源网格 OFDM 导频信号（或参考信号）；

3）用 IFFT 生成 OFDM 信号完全将时域信号通过资源网格分布与频域向对应。

我们随后回顾了接收端的典型组件，包括：

1）计算接收资源网格的 OFDM 接收器；

2）用参考信号进行信道估计；

3）通过内插方法计算全网格信道响应；

4）基于信道响应估计的频域均衡，用于恢复发射资源元素的最优估计。

最后，我们将所有收发端组件的 MATLAB 模型集成构建 LTE 标准单天线下行链路传输。另一方面，如 LTE 传输模式 1 定义，收发端进行 SISO 和 SIMO 下行链路处理。通过仿真，我们定量评估并测量了 BER 性能。结果显示收发端有效克服了多径衰落造成的码间串扰效应。在下一章中，我们将会介绍 MIMO 多天线方案，在发射端引入多天线。

参 考 文 献

[1] Y.S. Cho, J.K. Kim, W.Y. Yang, C.G. Kang, *MIMO-OFDM Wireless Communications with MATLAB*, John Wiley and Sons (Asia) Pte Ltd, 2010.

[2] 3GPP (2011) Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation Version 10.0.0. TS 36.211, January 2011.

[3] A. Ghosh, R. Ratasuk, *Essentials of LTE and LTE-A*, Cambridge University Press, 2011.

[4] H. Jafarkhani, *Space-Time Coding; Theory and Practice*, Cambridge University Press, 2005.

6　MIMO

迄今为止，我们研究了 LTE 标准中的调制、绕码、编码、信道模型和多载波传输。在本章中，我们将关注多天线传输。LTE 和 LTE - Advanced 标准只要依靠多天线集合或 MIMO（多输入多输出）技术实现了最大数据速率。LTE 可以认为是一种 MIMO - OFDM（正交频分复用）系统，即使用 MIMO 多天线配置实现 OFDM 多载波传输方案。

通常情况下，多天线传输方案映射调制数据符号到多天线端口。在 OFDM 传输方案中，每个天线构建资源网格，生成 OFDM 符号，并传输信号。在一个 MIMO - OFDM 系统中，资源网格映射处理和 OFDM 调制过程重复遍历每个天线。根据 MIMO 模式的不同，多天线扩展会提升数据速率或提升链路质量。

在本章中，我们首先回顾 LTE 四种传输模式中的 MIMO 算法。这些传输模式体现两种主要 MIMO 技术：发射分集（如空 - 频区块编码，SFBC）和使用或不使用延迟分集编码的空分复用。如我们前文所述，发射分集技术提升链路质量和可靠性但对数据速率或系统频谱利用率无帮助。另一方面，空分复用可以大幅度提高数据速率。

6.1　MIMO 定义

“MIMO 天线技术”通常被认为是指所有使用多发射和多接受天线通信的技术。LTE 标准集成 MIMO 多天线技术和 OFDM 多载波技术。实质上，在 LTE 中，多发射和多接收天线之间关系用每个独立子载波来说明比用全带宽整体特性说明更方便。图 6.1 所示为发射天线和接收天线的关系，并标注了每一组天线的信道增益。

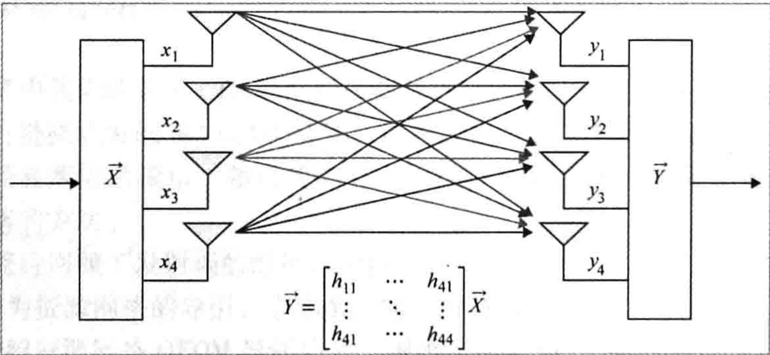


图 6.1　MIMO 发射端、接收端，和信道的结构图

在每个子载波上，不同天线间接收和发射的资源元素的关系由线性方程表示。在这个系统中，接收天线上接收资源元素向量由 MIMO 信道矩阵和发射天线上发射资源元素矩阵相乘得到。通过 MIMO 系统方程描述我们可以看到，为了在给定子载波上复原发射资源元素的最好估计，我们不仅需要接收资源元素向量也需要连接每个发射和接收天线的信道响应（或者 CSI、信道状态信息）。

6.2 MIMO 的动机

理论上，通信链路提升数据速率的最好途径是对给定发射功率情况下提升整体接收功率[1]。有效提升接收功率的办法是在接收端和/或发射端扩增天线。此类技术即多天线或 MIMO 技术。大幅改进吞吐量和比特误码率（BER）的 MIMO 技术刺激了多天线无线系统的发展。伴随着这些优势，可计算复杂度增加的劣势也凸显出来。MIMO 技术的复杂度一般随使用天线数量成指数增长。

各种 MIMO 技术中，空分复用引入多天线方法可以随天线数量线性提升吞吐量[1]。给定一种提高吞吐量的一般方法如提高功率只能成对数趋势改进性能，而使用 MIMO 技术大幅度提高吞吐量则反映了无线通信发展的历史性一步。

6.3 MIMO 的种类

LTE 吸取了 MIMO 大量优势，如在 9 个下行链路传输模式中使用多天线技术。LTE - Advanced 将发射天线数量扩展到八个。

我们现在来考察 MIMO 系统的数学基础。一个系统能否正确配置执行与接收端求解线性方程是否正确复原发射数据息息相关。在信道劣化的情况下，广域频谱上存在频率选择性相应。在每个子频带上，信道相应更接近平带并保持一定的增益。在 MIMO 系统内，每个字频带上发射和接收符号的关系可以表示成一个简单的增益值。这意味着多路收发天线间的关系可以表示成 MIMO 系统线性方程，它可以在全频谱中的任意子载波上求解以复原发射信号。

LTE 使用的 MIMO 算法可以细分为四类：接收端合并、发射分集、波束赋形和空分复用。我们会在本章下面的部分中简短讨论其中的三种技术。

6.3.1 接收端合并技术

接收端合并技术在接收端合并不同情况的发射信号以提升性能。该技术已在 3G 通信标准和 WiFi 以及 WiMAX 系统中广泛应用。接收端可使用两种合并方法：最大比合并（MRC）和选择式合并（SC）[2]。使用 MRC 情况下，我们合并多路接收信号（通常对它们取平均值）得到发射信号的最大似然估计。在使用 SC 情

况下，不会像 MRC 那样复杂，我们只使用接收写好和最高 SNR（信噪比）进行发射信号估计。

6.3.2 发射分集

在发射分集中，冗余信息在每个子载波不同的天线上发射。在此模式下，LTE 提高通信链路稳健性但不会提高数据速率。发射分集术语多天线技术中空－时编码的范围。空－时码能够实现分集阶数等于接收天线和发射天线数成绩。SFBC，该技术与空－时分组编码（STBC）类似，是一种应用于 LTE 中的发射分集技术。

6.3.3 空分复用

在空分复用中，系统在不同天线上传输独立（非冗余）信息。该 MIMO 模式可以在给定通信链路上与发射天线数量成比例大幅的提高数据速率。空分复用传输独立数据流的能力伴随着代价。不过，空分复用可以克服 MIMO 方程矩阵秩不足的问题。LTE 空分复用引入多种技术减小秩不足出现的概率以发挥优势。

6.4 MIMO 的覆盖范围

在本书中，我们关注 MIMO 传输前四种模式的信号处理。波束赋形，用于模式 6，与多播有关，是协作多点传输的关键。多用户 MIMO（MU－MIMO），用于模式 5 和模式 7 ~9，可以理解为是模式 3 和模式 4 单用户配置的扩展。我们将在其他部分更深入研究有关下行链路与上行链路波束赋形以及 MU－MIMO。

6.5 MIMO 信道

MIMO 信道定义了在多路发射天线发射信号和多路接收天线接收信号之间的关系。链路数量等于发射天线数（numTx）和接收天线数（numRx）的乘积。

在平坦衰落情况下，一个时间点上任意给定收发天线对之间的关系可以由一个标量增益值，即信道路径增益确定。这些信道增益的集合定义了信道矩阵 $\boldsymbol{H}$。信道矩阵的阶数等于（numTx，numRx）。一个系统的线性方程表述了每个接收天线的接收信号、每个发射天线的发射信号，以及信道矩阵的关系。图 6.2 表示了 2×2 MIMO 信道平坦衰落响应条件下 $X(n)$（采样时间 n 的发射向量），$Y(n)$（采样时间 n 的接收向量），以及 $\boldsymbol{H}(n)$（采样时间 n 的信道矩阵）之间的关系。

时序 $n=1$，…，nSamp，nSamp 为一个天线上每个子帧传输符号数量。所有子帧发射信号的阶数为（nSamp，numTx），接收信号的阶数为（nSamp，num-

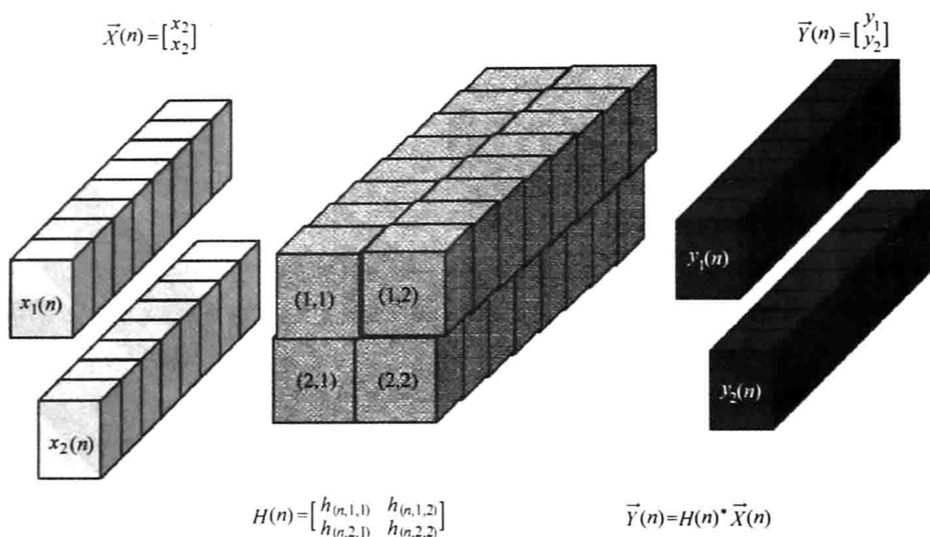


图 6.2　一个 2×2 MIMO 平坦衰落响应信道

Rx)，信道矩阵为一个阶数是（nSamp，numTx，numRx）的 3D 矩阵。

在多径衰落情况下，一个时间点上任意给定收发天线对的关系可以由信道 - 路径增益向量表示。一个时间点上每个接受信号取决于发射信号过去和当前值。另外一个参数需要被引入：路径延迟 L。为了计算多径情况下的接收信号，必须对每个路径延迟向量值重复平坦衰落情况下的 MIMO 处理。

因此，全子帧传输信号的阶数为（nSamp，numTx），接收信号的阶数为（nSamp，numRx），但信道矩阵为（nSamp，L，numTx，numRx）阶数的 4D 矩阵。图 6.3 表示了 2×2 MIMO 多径衰落响应信道条件下发射信号 X（n），接收信号 Y（n），信道矩阵 $\boldsymbol{H}$（n，k）的关系。时序 $n=1$，…，nSamp，nSamp 定义同上。路径延迟序数 $k=1$，…L，L 为路径延迟数。

6.5.1 MATLAB 实现

我们可以使用 comm. MIMOChannel 系统对象研究多天线已经多传播路径效应以及配置 MIMO 信道模型。

comm. MIMOChannel 系统对象用诸如收发天线数、延迟参数、多普勒频移对平坦或频率选择性 MIMO 信道进行动态建模。

下面的 MATLAB 函数表示一个既可以表征平坦也可以表征频率选择性的 MIMO 衰落信道模型。该函数以 2D 矩阵为输入变量（x）。矩阵第一个阶数

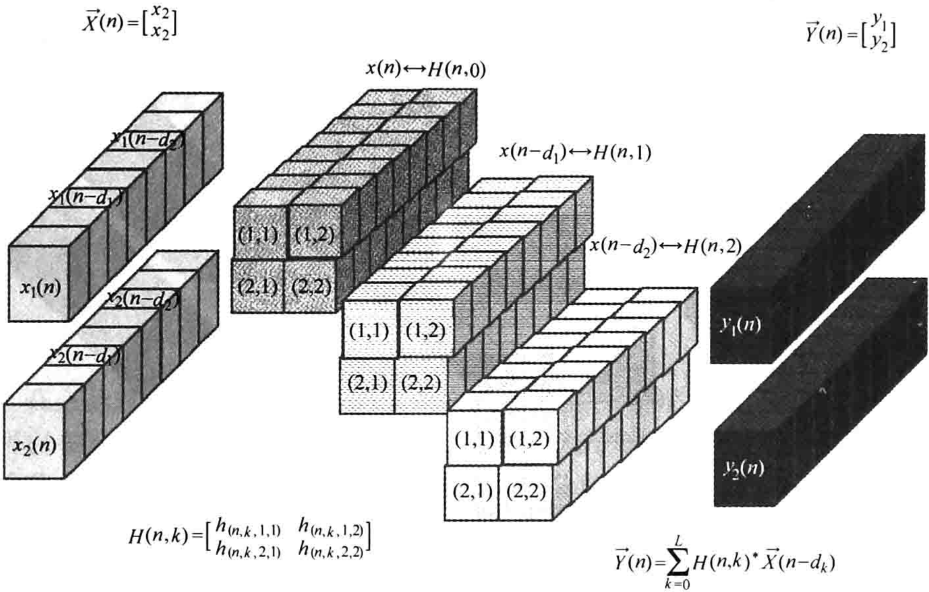


图 6.3　一个 2×2 MIMO 多径衰落响应信道

（nSamp）为一个子帧每个发射天线采样数。第二个阶数为发射天线数量（numTx）。函数有两个输出变量。第一个（y）是输入变量（x）经过衰落信道后的版本。输出信号的第一个阶数与输入信号第一个阶数（nSamp）相同。第二个阶数等于接收天线数（numRx）。函数的第二个输出为表示信道矩阵的多维矩阵（$\boldsymbol{H}$）（也就是路径增益）。路径增益作用于输入变量（x）生成输出衰落信号（y）。

Algorithm

MATLAB function

```
function [y, yPg] = MIMOFadingChan(in, prmLTE, prmMdl)
% MIMOFadingChan
%#codegen
% Get simulation params
numTx         = prmLTE.numTx;
numRx         = prmLTE.numRx;
chanSRate     = prmLTE.chanSRate;
chanMdl       = prmMdl.chanMdl;
corrLvl       = prmMdl.corrLevel;
PathDelays    = prmMdl.PathDelays ;
PathGains     = prmMdl.PathGains ;
```

```
Doppler      = prmMdl.Doppler;
ChannelType = prmMdl.ChannelType ;
AntConfig     = prmMdl.AntConfig;
% Initialize objects
persistent chanObj;
if isempty(chanObj)
   if ChannelType ==1
      chanObj = comm.MIMOChannel('SampleRate', chanSRate, ...
         'MaximumDopplerShift', Doppler, ...
         'PathDelays', PathDelays,...
         'AveragePathGains', PathGains,...
         'RandomStream', 'mt19937ar with seed',...
         'Seed', 100,...
         'NumTransmitAntennas', numTx,...
         'TransmitCorrelationMatrix', eye(numTx),...
         'NumReceiveAntennas', numRx,...
         'ReceiveCorrelationMatrix', eye(numRx),...
         'PathGainsOutputPort', true,...
         'NormalizePathGains', false,...
         'NormalizeChannelOutputs', true);
   else
      chanObj = comm.LTEMIMOChannel('SampleRate', chanSRate, ...
         'Profile', chanMdl, ...
         'AntennaConfiguration', AntConfig, ...
         'CorrelationLevel', corrLvl,...
         'RandomStream', 'mt19937ar with seed',...
         'Seed', 100,...
         'PathGainsOutputPort', true);
   end
end
[y, yPg] = step(chanObj, in);
```

在这个函数中，我们用两种不同的系统对象描述 MIMO 信道操作。comm. MIMOChannel 系统对象为一般 MIMO 信道的模型。它使用路径延迟、路径增益以及多普勒频移等定义模型。

comm. LTEMIMOChannel 系统对象为特定 LTE 信道模型。我们将在下一节详细讨论它。这个系统对象由不同的参数设置，如天线配置和发射天线间相关性水平等，用于计算信道建模所必需的全需条件。该函数实现的 MIMO 衰落特性遵循 LTE 标准规定[3]。

6.5.2 LTE 特征信道模型

3GPP 技术推荐（TR）36.104[3] 定义了三种不同的多径衰落信道模型：扩展步行 A 信道模型（EPA）、扩展车载 A 信道模型（EVA），和扩展典型城市信道模型（ETU）。本书中使用的信道建模函数基于这三种模型构建。因闭环空分

复用只能用于高数据速率低移动率的情况下，我们将不会涉及高移动率配置。

结合前文中描述的一般信道模型，该模型可以满足我们在各种参考信道条件下评估收发器性能。

多径衰落信道模型由延迟参数与最大多普勒频率共同确定。信道模型的延迟参数对应低、中，和高三种延迟扩散环境，对应最大多普勒频移为 5、70，300Hz。表 6.1 总结了由每个信道模型的多径延迟值（ns）和相对功率（dB）表征的延迟参数。

在 MIMO 传输模式下，收发天线间的空间相关性是影响整体系统性能的重要参数。MIMO 需要在最大散射和多径衰落环境下很好地工作。因此，它需要最小化收发端多路天线端口间的相关性。MIMO 将信道矩阵秩不足的概率降到最小并提升性能。

表 6.1 LTE 信道模型（EPA，EVA，ETU）：**延迟特性**

信道模型	多径延迟/ns	相对功率/dB
扩展步行 A（EPA）	[0 30 70 90 110 190 410]	[0 -1 -2 -3 -8 -17.2 -20.8]
扩展车载 A（EVA）	[0 30 150 310 370 710 1090 1730 2510]	[0 -1.5 -1.4 -3.6 -0.6 -9.1 -7 -12 -16.9]
扩展典型城市（ETU）	[0 50 120 200 230 500 1600 2300 5000]	[-1 -1 -1000 -3 -5 -7]

例如，对 2×2 MIMO 天线配置，发射端（eNodeB）空间相关性矩阵（$\boldsymbol{M}_{\text{tx}}$）为 2×2 矩阵，它的对角元素为 1 而非对角元素为参数（α），即 $\boldsymbol{M}_{\text{tx}}=\begin{bmatrix}1 & \alpha\\ \alpha^{*} & 1\end{bmatrix}$。与此相似，在接收端（UE）空间相关性矩阵（$\boldsymbol{M}_{\text{rx}}$）也为 2×2 矩阵，它的非对角元素为参数（$\beta$），即 $\boldsymbol{M}_{\text{rx}}=\begin{bmatrix}1 & \beta\\ \beta^{*} & 1\end{bmatrix}$。注意当这两个参数为实数时，不需要求其共轭。

在 4×4 天线配置情况下，收发端空间相关性矩阵由参数 α 和 β 同时表示。发射端（eNodeB）空间相关性矩阵为 4×4 矩阵，即

$$\boldsymbol{M}_{tx}=\begin{bmatrix}1 & \alpha^{\frac{1}{9}} & \alpha^{\frac{4}{9}} & \alpha\\ \alpha^{\frac{1}{9}} & 1 & \alpha^{\frac{1}{9}} & \alpha^{\frac{4}{9}}\\ \alpha^{\frac{4}{9}} & \alpha^{\frac{1}{9}} & 1 & \alpha^{\frac{1}{9}}\\ \alpha & \alpha^{\frac{4}{9}} & \alpha^{\frac{1}{9}} & 1\end{bmatrix}.$$

LTE 定义了三个不同的相关性水平：低（无相关性）、中和高。三种相关性水平由相关性矩阵的参数值（α 和 β）反映，见表 6.2。

表 6.2 LTE 信道模型：相关性水平和空间相关性矩阵系数

LTE MIMO 信道相关水平	α	β
低相关	0	0
中等相关	0.3	0.9
高相关	0.9	0.9

6.5.3 MATLAB 实现

系统对象 comm.LTEMIMOChannel 定义 LTE 信道模型并实现三种类型的信道（EPA、EVA 和 ETU）如前章节所述。该系统对象由不同的参数设置，如天线配置和发射天线间相关性水平等，用于计算信道建模所必需的全部条件。该系统对象实现的 MIMO 衰落特性遵循 LTE 技术推荐 36.104 规定[3]。

因为该系统对象为 MATLAB 管理系统对象，我们可以用命令 edit comm.LTEMIMOChannel 研究 MATLAB 代码实现过程。如各种 LTE 信道模型的延迟特性，可以由系统对象的 setDelayDopplerProfiles 函数内数行 MATLAB 代码实现。

Algorithm

MATLAB code segment

```
function setDelayDopplerProfiles(obj)
   EPAPathDelays = [0 30 70 90 110 190 410]*1e-9;
   EPAPathGains  = [0 -1 -2 -3 -8 -17.2 -20.8];
   EVAPathDelays = [0 30 150 310 370 710 1090 1730 2510]*1e-9;
   EVAPathGains  = [0 -1.5 -1.4 -3.6 -0.6 -9.1 -7 -12 -16.9];
   ETUPathDelays = [0 50 120 200 230 500 1600 2300 5000]*1e-9;
   ETUPathGains  = [-1 -1 -1 0 0 0 -3 -5 -7];
   switch obj.Profile
   case 'EPA 5Hz'
      obj.PathDelays          = EPAPathDelays;
      obj.AveragePathGains    = EPAPathGains;
      obj.MaximumDopplerShift = 5;
   case 'EVA 5Hz'
      obj.PathDelays          = EVAPathDelays;
      obj.AveragePathGains    = EVAPathGains;
      obj.MaximumDopplerShift = 5;
   case 'EVA 70Hz'
      obj.PathDelays          = EVAPathDelays;
      obj.AveragePathGains    = EVAPathGains;
      obj.MaximumDopplerShift = 70;
   case 'ETU 70Hz'
      obj.PathDelays          = ETUPathDelays;
      obj.AveragePathGains    = ETUPathGains;
      obj.MaximumDopplerShift = 70;
   case 'ETU 300Hz'
      obj.PathDelays          = ETUPathDelays;
      obj.AveragePathGains    = ETUPathGains;
      obj.MaximumDopplerShift = 300;
   end
```

6.5.4 MIMO 信道初始化

随着我们初始化仿真，多个静态和多函数复用的属性存储于各个仿真参数结构体中。在第 4 章中，我们引入结构体参数 prmLTEDLSCH，包含执行 Turbo 编码和载荷符号生成的参数，包括资源网格映射、OFDM 信号生成，和 MIMO 处理。在本章中，我们引入结构体参数 prmMdl，包含多个与 MIMO 衰落信道和仿真终止条件相关的属性。

下面的 MATLAB 函数初始化参数结构体 prmMdl。根据仿真开始时定义的 9 个参数值，函数设置不同的结构字段。例如，根据 chanMdl 输入字符串不同，路径延迟、路径增益、多普勒频移，和信道类型也不相同。该参数设置决定了衰落类型、以及多普勒频移参数反映的移动率对衰落造成的影响。

Algorithm

MATLAB function

```
function prmMdl = prmsMdl(chanSRate, chanMdl, numTx, numRx, ...
   corrLvl, chEstOn, snrdB, maxNumErrs, maxNumBits)
prmMdl.chanMdl = chanMdl;
prmMdl.AntConfig=char([48+numTx,'x',48+numRx]);
switch chanMdl
   case 'flat-low-mobility',
      prmMdl.PathDelays = 0*(1/chanSRate);
      prmMdl.PathGains  = 0;
      prmMdl.Doppler=0;
      prmMdl.ChannelType =1;
   case 'flat-high-mobility',
      prmMdl.PathDelays = 0*(1/chanSRate);
      prmMdl.PathGains  = 0;
      prmMdl.Doppler=70;
      prmMdl.ChannelType =1;
   case 'frequency-selective-low-mobility',
      prmMdl.PathDelays = [0 10 20 30 100]*(1/chanSRate);
      prmMdl.PathGains  = [0 -3 -6 -8 -17.2];
      prmMdl.Doppler=0;
      prmMdl.ChannelType =1;
   case 'frequency-selective-high-mobility',
      prmMdl.PathDelays = [0 10 20 30 100]*(1/chanSRate);
      prmMdl.PathGains  = [0 -3 -6 -8 -17.2];
      prmMdl.Doppler=70;
      prmMdl.ChannelType =1;
   case 'EPA 0Hz'
      prmMdl.PathDelays = [0 30 70 90 110 190 410]*1e-9;
      prmMdl.PathGains  = [0 -1 -2 -3 -8 -17.2 -20.8];
      prmMdl.Doppler=0;
      prmMdl.ChannelType =1;
```

```
    otherwise
        prmMdl.PathDelays = 0*(1/chanSRate);
        prmMdl.PathGains  = 0;
        prmMdl.Doppler=0;
        prmMdl.ChannelType =2;
end
prmMdl.corrLevel = corrLvl;
prmMdl.chEstOn = chEstOn;
prmMdl.snrdB=snrdB;
prmMdl.maxNumBits=maxNumBits;
prmMdl.maxNumErrs=maxNumErrs;
```

6.5.5 添加 AWGN

在第 8 章中，我们会介绍 AWGNchannel 函数。它向信号添加白高斯噪声。下面的 MATLAB 代码片段表现了信道模型如何体现衰落信道和 AWGN（加性白高斯噪声）信道。首先，程序调用 MIMOFadingChan 函数，我们生成发射信号衰落版本（rxFade）和相应的信道矩阵（chPathG）。注意在 MIMOFadingChan 函数中我们定义路径增益为 1。如忽略这个定义，因 MIMO 衰落信道以现行合并多发射天线计算衰落信号，输出信号（rxFade）将无法统一方差。为了计算 AWGN-Channel 函数需要的噪声方差，我们必须首先计算信号方差（sigPow）并由信号功率与 SNR 值得到分贝表示的噪声方差。

Algorithm

MATLAB code segment

```
%% Channel
% MIMO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
% Add AWG noise
sigPow = 10*log10(var(rxFade));
nVar = 10.^(0.1.*(sigPow-snrdB));
rxSig =  AWGNChannel(rxFade, nVar);
```

最后，通过分贝 - 线性变换我们计算得到由线性向量表示的噪声方差（nVar）。因衰落输出信号（rxFade）的第二阶等于接收天线数（numRx），噪声方差向量的阶数也等于接收天线数。我们很快会看到，噪声方差估计是均衡和解调过程中重要的参数。

6.6 MIMO 的一般特征

前面的章节介绍的一些多载波传输的功能组件在本章中需要更新以满足多天

线传输应用。这些功能组件包括资源元素映射和反映射、信道估计方法、信道响应提取，和均衡。另一方面，一些功能组件为 MIMO 特有，包括预编码、层映射，和 MIMO 接收器。在本节中我们详细更新所需的一般组件并介绍 MIMO 特有的处理。

6.6.1 MIMO 资源网格结构

小区专有参考（CSR）信号在频域均衡（见第 5 章）和 MIMO 接收器处理（将会在下文介绍）中发挥关键作用。但 CSR 的在 MIMO 中因多天线技术的存在有着本质的不同。当一个 CSR 在任意天线的任意给定子载波上发送时，所有其他天线必须在相同子载波上传输空信号（零值信号）。这就要求在资源网格中设置一个新的组件，它称为频谱零值。

图 6.4 表示了使用 1 个、2 个，和 4 个天线情况下 CSR 和频谱零值在普通资源块内的位置。嘴上方为单天线情况，CSR 在每个子帧内使用四个 OFDM 符号且每个符号内有两个 CSR 采样。在这种情况下，因为只有有一个天线传输信息，所以不需要频谱零值。该配置下进行的资源元素映射和反映射，如上一章函数 REmapper_ITx. m 和 REdemapper_ITx. m 所示。

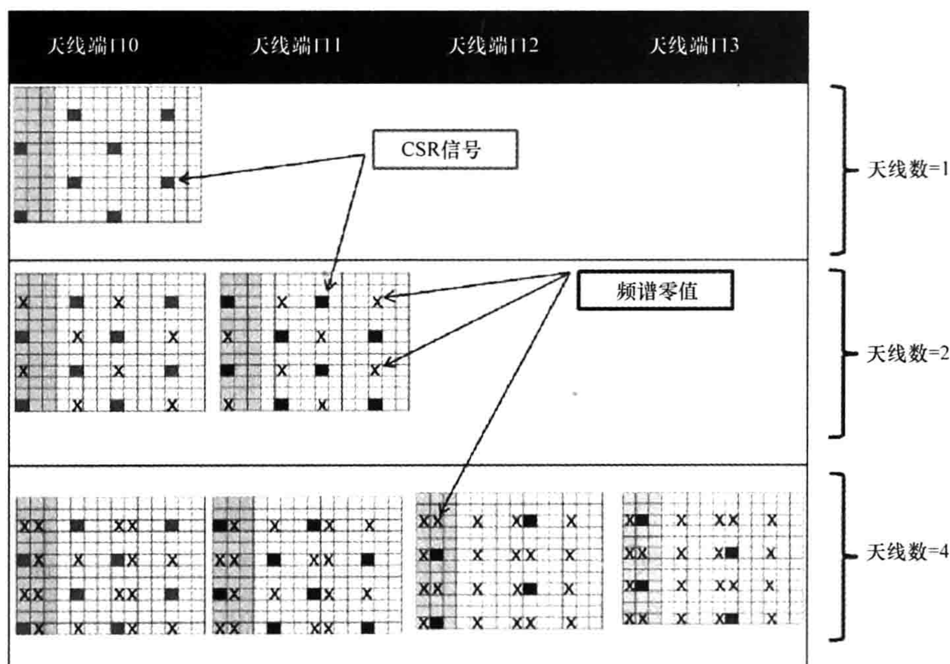


图 6.4　1 个、2 个和 4 个天线配置的小区特有参考（CSR）信号和频谱零值

在 2×2 MIMO 配置下，即图 6.4 中间部分所示，我们可以发现每个天线端口都出现了频谱零值（零值资源元素，由×符号表示）。注意一个天线端口频谱零值的位置即另一个天线端口 CSR 的位置。这意味着两个天线有 4 个包含 CSR 信号的 OFDM 符号，每个资源块内每个符号含两个 CSR 信号。

在 4×4 配置中，如图 6.4 下部所示，我们可以看到两点不同：

1）第一和第二个天线的 CSR 密度 7 与第三个第四个天线不同；

2）四个天线的频谱零值比两天线配置高。

第一个第二个天线的 CSR 信号密度与 2×2 MIMO 配置相同。这意味着两个天线有 4 个包含 CSR 信号的 OFDM 符号，每个资源块内每个符号含两个 CSR 信号，第 3 个和第 4 个天线，只有两个 OFDM 符号包含 CSR 信号，它们位于第 1 个和第 8 个符号上，每个资源块含两个 CSR 信号。任意一个发射天线上频谱零值的位置刚好为其他发射天线上 CSR 信号的位置。因此，CSR 与频谱零值的和对所有发射天线为常数。

多天线情况下 CSR 信号和频谱零值位置与资源元素映射与反映射有关。下面我们会讨论实现映射与反映射功能的函数：REmapper_mTx.m 和 REdemapper_mTx.m

6.6.2 资源元素映射

在本节中，我们详细讲解 MIMO 传输模式的资源元素映射。如单天线传输，资源元素映射过程实质上是生成指向资源网格矩阵的索引并将多种信息类型放置于网格的过程。LTE 下行链路资源网格的信号类型包括用户数据（PDSCH，物理下行链路公共信道）、CSR 信号、主同步信号和辅助同步信号（PSS 和 SSS），物理广播信道（PBCH）和物理下行链路控制信道（PDCCH）。除了我们需要引入两个特性之外，MIMO 资源网格的构成与单天线情形类似。这两个特性分别为：首先，需要实现频谱零值以在频谱估计中减小 CSR 信号串扰。其次，实现 CSR 在 4×4 配置中的特殊位置，即 CSR 随天线不同而位置不同。

下面的 MATLAB 函数描述了资源元素映射。函数实现了 SISO 映射、SIMO 映射，和 MIMO 情况下 1、2、4 天线配置的映射。函数输入为用户数据（in）、CSR 信号（csr）、子帧索引（nS），和结构体 prmLTEPDSCH 中的 PDSCH 参数。根据 BCH（广播信道）、SSS、PSS，和 DCI（下行链路控制信息）的不同类型，函数有额外的输入。输出变量（y）为资源网格矩阵。资源网格为一个 3D 矩阵，它的第一阶是子载波数，第二阶等于每个子帧的 OFDM 符号数，第三阶是发射天线数。函数由三部分构成。第一部分，根据发射天线数（numTx）不同，我们初始化索引用户数据（idx_data），CSR 信号（idx_csr），和 DCI（idx_pdcch）。第二部分，我们使用函数 ExpungeFrom.m 计算除 CSR 之外的用户数据索引。我

们根据子帧索引（nS）值从用户数据和 DCI 中空出 PSS、SSS，和 PBCH 的位置。最后，第三部分，我们初始化输出缓冲区。通过将全部资源网格初始化清空我们实质上已将频谱零值放置在没有任何其他信息写入的位置上。对于每个传输天线我们可以用第一第二部分生成的索引填充资源网格。

Algorithm

MATLAB function

```
function y = REmapper_mTx(in, csr, nS, prmLTE, varargin)
%#codegen
switch nargin
   case 4, pdcch=[];pss=[];sss=[];bch=[];
   case 5, pdcch=varargin{1};pss=[];sss=[];bch=[];
   case 6, pdcch=varargin{1};pss=varargin{2};sss=[];bch=[];
   case 7, pdcch=varargin{1};pss=varargin{2};sss=varargin{3};bch=[];
   case 8, pdcch=varargin{1};pss=varargin{2};sss=varargin{3};bch=varargin{4};
   otherwise
      error('REMapper has 4 to 8 arguments!');
end
% NcellID = 0;                          % One of possible 504 values
% Get input params
numTx            = prmLTE.numTx;         % Number of transmit antennas
Nrb              = prmLTE.Nrb;
Nrb_sc           = prmLTE.Nrb_sc;        % 12 for normal mode
Ndl_symb         = prmLTE.Ndl_symb;      % 7    for normal mode
numContSymb    = prmLTE.contReg;         % either {1, 2, 3}
%% Specify resource grid location indices for CSR, PDCCH, PDSCH, PBCH, PSS, SSS
coder.varsize('idx_data');
lenOFDM = Nrb*Nrb_sc;
ContREs=numContSymb*lenOFDM;
idx_dci=1:ContREs;
lenGrid= lenOFDM * Ndl_symb*2;
idx_data  = ContREs+1:lenGrid;
%% 1st: Indices for CSR pilot symbols
idx_csr0   = 1:6:lenOFDM;          % More general starting point = 1+mod(NcellID, 6);
idx_csr4   = 4:6:lenOFDM;          % More general starting point = 1+mod(3+NcellID, 6);
% Depends on number of transmit antennas
switch numTx
   case 1
      idx_csr     = [idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0, 11*lenOFDM
+idx_csr4];
      idx_data   = ExpungeFrom(idx_data,idx_csr);
      idx_pdcch = ExpungeFrom(idx_dci,idx_csr0);
      idx_ex      = 7.5* lenOFDM - 36 + (1:6:72);
      a=numel(idx_csr); IDX=[1, a];
```

```
  case 2
    idx_csr1     = [idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0, 11*lenOFDM
+idx_csr4];
    idx_csr2     = [idx_csr4, 4*lenOFDM+idx_csr0, 7*lenOFDM+idx_csr4, 11*lenOFDM
+idx_csr0];
    idx_csr      = [idx_csr1, idx_csr2];
    % Exclude pilots and NULLs
    idx_data   = ExpungeFrom(idx_data,idx_csr1);
    idx_data   = ExpungeFrom(idx_data,idx_csr2);
    idx_pdcch  = ExpungeFrom(idx_dci,idx_csr0);
    idx_pdcch  = ExpungeFrom(idx_pdcch,idx_csr4);
    idx_ex       = 7.5* lenOFDM - 36 + (1:3:72);
    % Point to pilots only
    a=numel(idx_csr1); IDX=[1, a; a+1, 2*a];
  case 4
    idx_csr1     = [idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0, 11*lenOFDM
+idx_csr4];
    idx_csr2     = [idx_csr4, 4*lenOFDM+idx_csr0, 7*lenOFDM+idx_csr4, 11*lenOFDM
+idx_csr0];
    idx_csr33   = [lenOFDM+idx_csr0, 8*lenOFDM+idx_csr4];
    idx_csr44   = [lenOFDM+idx_csr4, 8*lenOFDM+idx_csr0];
    idx_csr      = [idx_csr1, idx_csr2, idx_csr33, idx_csr44];
    % Exclude pilots and NULLs
    idx_data   = ExpungeFrom(idx_data,idx_csr1);
    idx_data   = ExpungeFrom(idx_data,idx_csr2);
    idx_data   = ExpungeFrom(idx_data,idx_csr33);
    idx_data   = ExpungeFrom(idx_data,idx_csr44);
    % From pdcch
    idx_pdcch  = ExpungeFrom(idx_dci,idx_csr0);
    idx_pdcch  = ExpungeFrom(idx_pdcch,idx_csr4);
    idx_pdcch  = ExpungeFrom(idx_pdcch,lenOFDM+idx_csr0);
    idx_pdcch  = ExpungeFrom(idx_pdcch,lenOFDM+idx_csr4);
    idx_ex       = [7.5* lenOFDM - 36 + (1:3:72), 8.5* lenOFDM - 36 + (1:3:72)];
    % Point to pilots only
    a=numel(idx_csr1); b=numel(idx_csr33);
    IDX =[1, a; a+1, 2*a; 2*a+1, 2*a+b; 2*a+b+1, 2*a+2*b];
  otherwise
    error('Number of transmit antennas must be {1, 2, or 4}');
end
%% 3rd: Indices for PDSCH and PDSCH data in OFDM symbols where pilots are present
%% Handle 3 types of subframes differently
switch nS
  %% 4th: Indices for BCH, PSS, SSS are only found in specific subframes 0 and 5
  % These symbols share the same 6 center sub-carrier locations (idx_ctr)
  % and differ in OFDM symbol number.
  case 0    % Subframe 0
    % PBCH, PSS, SSS are available + CSR, PDCCH, PDSCH
```

```
        idx_ctr = 0.5* lenOFDM - 36 + (1:72) ;
        idx_SSS  = 5* lenOFDM + idx_ctr;
        idx_PSS  = 6* lenOFDM + idx_ctr;
        idx_bch0=[7*lenOFDM + idx_ctr, 8*lenOFDM + idx_ctr, 9*lenOFDM + idx_ctr,
10*lenOFDM + idx_ctr];
        idx_bch = ExpungeFrom(idx_bch0,idx_ex);
        idx_data  = ExpungeFrom(idx_data,[idx_SSS, idx_PSS, idx_bch]);
    case 10  % Subframe 5
        % PSS, SSS are available + CSR, PDCCH, PDSCH
        % Primary and Secondary synchronization signals in OFDM symbols 5 and 6
        idx_ctr = 0.5* lenOFDM - 36 + (1:72) ;
        idx_SSS  = 5* lenOFDM + idx_ctr;
        idx_PSS  = 6* lenOFDM + idx_ctr;
        idx_data  = ExpungeFrom(idx_data,[idx_SSS, idx_PSS]);
    otherwise % other subframes
        % Nothing to do
end
% Initialize output buffer
y = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2, numTx));
for m=1:numTx
    grid = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2));
    grid(idx_data.')=in(:,m);                          % Insert user data
    Range=idx_csr(IDX(m,1):IDX(m,2)).';                % How many pilots in this antenna
   csr_flat=packCsr(csr, m, numTx);                  % Pack correct number of CSR values
    grid(Range)=  csr_flat(:);                         % Insert CSR pilot symbols
    if ~isempty(pdcch), grid(idx_pdcch)=pdcch(:,m);end
% Insert Physical Downlink Control Channel (PDCCH)
    if ~isempty(pss),    grid(idx_PSS)=pss(:,m);end
% Insert Primary Synchronization Signal (PSS)
    if ~isempty(sss),    grid(idx_SSS)=sss(:,m);end
% Insert Secondary Synchronization Signal (SSS)
    if ~isempty(bch),    grid(idx_bch)=bch(:,m);end % Insert Broadcast Channel data (BCH)
    y(:,:,m)=grid;
end
end
%% Helper function
function csr_flat=packCsr(csr, m, numTx)

    if ((numTx==4)&&(m>2))                            % Handle special case of 4Tx
      csr_flat=csr(:,[1,3],m);                        % Extract pilots in this antenna
    else
        csr_flat=csr(:,:,m);
    end
end
```

6.6.3 资源元素反映射

资源元素反映射是指反向资源网格映射的处理过程。下面的 MATLAB 函数

描述了参考信号和数据如何从接收端恢复的资源网格中提取出来。函数有三个变量声明：恢复的资源网格（in）、子帧索引（nS）和 PDSCH 参数设置。函数输出提取用户数据（data）、用户数据索引（idx_data）以及 CSR 信号（csr）和 DCI（pdcch）（可选），主同步信号和辅助同步信号（pss，sss）以及 BCH 信号（bch）。因不同子帧包含不同的内容，第二个输入子帧索引参数（nS）可以使函数分配正确数据。生成反映射中生成索引的算法与资源映射中相同。在多天线情况下，资源网格输入是一个 3D 矩阵。前两个阶数由每个接收天线资源网格大小确定，第三个阶数为接收天线数。与资源映射函数相似，资源反映射函数也有三个部分。在其前两个部分，我们计算资源网格各个组件的索引。它们包括索引用户数据（idx_data）、CSR 信号（idx_csr）、DCI（idx_pdcch）、主同步和辅助同步信号（idx_PSS，idx_SSS）以及 BCH 信号（idx_bch）。在第三部分，我们根据第一和第二部分生成的索引从资源网格提取相应的数据。

Algorithm

MATLAB function

```
function [data, csr, idx_data, pdcch, pss, sss, bch] = REdemapper_mTx(in, nS, prmLTE)
%#codegen
% NcellID = 0;                          % One of possible 504 values
% Get input params
numTx          = prmLTE.numTx;          % number of receive antennas
numRx          = prmLTE.numRx;          % number of receive antennas
Nrb            = prmLTE.Nrb;            % either of {6,...,100 }
Nrb_sc         = prmLTE.Nrb_sc;         % 12 for normal mode
Ndl_symb       = prmLTE.Ndl_symb;       % 7    for normal mode
numContSymb    = prmLTE.contReg;        % either {1, 2, 3}
Npss           = prmLTE.numPSSRE;
Nsss           = prmLTE.numSSSRE;
Nbch           = prmLTE.numBCHRE;
%% Specify resource grid location indices for CSR, PDCCH, PDSCH, PBCH, PSS, SSS
coder.varsize('idx_data');
coder.varsize('idx_dataC');
lenOFDM = Nrb*Nrb_sc;
ContREs=numContSymb*lenOFDM;
idx_dci=1:ContREs;
lenGrid= lenOFDM * Ndl_symb*2;
idx_data  = ContREs+1:lenGrid;
%% 1st: Indices for CSR pilot symbols
idx_csr0  = 1:6:lenOFDM;        % More general starting point = 1+mod(NcellID, 6);
idx_csr4  = 4:6:lenOFDM;        % More general starting point = 1+mod(3+NcellID, 6);
% Depends on number of transmit antennas
switch numTx
   case 1
      idx_csr    = [idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0, 11*lenOFDM
+idx_csr4];
```

```
        idx_data   = ExpungeFrom(idx_data,idx_csr);
        idx_pdcch = ExpungeFrom(idx_dci,idx_csr0);
        idx_ex      = 7.5* lenOFDM - 36 + (1:6:72);
    case 2
        idx_csr1    = [idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0, 11*lenOFDM
+idx_csr4];
        idx_csr2    = [idx_csr4, 4*lenOFDM+idx_csr0, 7*lenOFDM+idx_csr4, 11*lenOFDM
+idx_csr0];
        idx_csr     = [idx_csr1, idx_csr2];
        % Exclude pilots and NULLs
        idx_data    = ExpungeFrom(idx_data,idx_csr1);
        idx_data    = ExpungeFrom(idx_data,idx_csr2);
        idx_pdcch  = ExpungeFrom(idx_dci,idx_csr0);
        idx_pdcch  = ExpungeFrom(idx_pdcch,idx_csr4);
        idx_ex       = 7.5* lenOFDM - 36 + (1:3:72);
    case 4
        idx_csr1    = [idx_csr0, 4*lenOFDM+idx_csr4, 7*lenOFDM+idx_csr0, 11*lenOFDM
+idx_csr4];
        idx_csr2    = [idx_csr4, 4*lenOFDM+idx_csr0, 7*lenOFDM+idx_csr4, 11*lenOFDM
+idx_csr0];
        idx_csr33  = [lenOFDM+idx_csr0, 8*lenOFDM+idx_csr4];
        idx_csr44  = [lenOFDM+idx_csr4, 8*lenOFDM+idx_csr0];
        idx_csr     = [idx_csr1, idx_csr2, idx_csr33, idx_csr44];
        % Exclude pilots and NULLs
        idx_data    = ExpungeFrom(idx_data,idx_csr1);
        idx_data    = ExpungeFrom(idx_data,idx_csr2);
        idx_data    = ExpungeFrom(idx_data,idx_csr33);
        idx_data    = ExpungeFrom(idx_data,idx_csr44);
        % From pdcch
        idx_pdcch  = ExpungeFrom(idx_dci,idx_csr0);
        idx_pdcch  = ExpungeFrom(idx_pdcch,idx_csr4);
        idx_pdcch  = ExpungeFrom(idx_pdcch,lenOFDM+idx_csr0);
        idx_pdcch  = ExpungeFrom(idx_pdcch,lenOFDM+idx_csr4);
        idx_ex       = [7.5* lenOFDM - 36 + (1:3:72), 8.5* lenOFDM - 36 + (1:3:72)];

    otherwise
        error('Number of transmit antennas must be {1, 2, or 4}');
end
%% 3rd: Indices for PDSCH and PDSCH data in OFDM symbols where pilots are present
%% Handle 3 types of subframes differently
switch nS
    %% 4th: Indices for BCH, PSS, SSS are only found in specific subframes 0 and 5
    % These symbols share the same 6 center sub-carrier locations (idx_ctr)
    % and differ in OFDM symbol number.
    case 0    % Subframe 0
        % PBCH, PSS, SSS are available + CSR, PDCCH, PDSCH
        idx_ctr = 0.5* lenOFDM - 36 + (1:72) ;
        idx_SSS  = 5* lenOFDM + idx_ctr;
        idx_PSS  = 6* lenOFDM + idx_ctr;
```

```
    idx_bch0=[7*lenOFDM + idx_ctr, 8*lenOFDM + idx_ctr, 9*lenOFDM + idx_ctr,
10*lenOFDM + idx_ctr];
    idx_bch = ExpungeFrom(idx_bch0,idx_ex);
    idx_data  = ExpungeFrom(idx_data,[idx_SSS, idx_PSS, idx_bch]);
  case 10 % Subframe 5
    % PSS, SSS are available + CSR, PDCCH, PDSCH
    % Primary and Secondary synchronization signals in OFDM symbols 5 and 6
    idx_ctr = 0.5* lenOFDM - 36 + (1:72) ;
    idx_SSS  = 5* lenOFDM + idx_ctr;
    idx_PSS  = 6* lenOFDM + idx_ctr;
    idx_data  = ExpungeFrom(idx_data,[idx_SSS, idx_PSS]);
  otherwise % other subframes
    % Nothing to do
end
%% Write user data PDCCH, PBCH, PSS, SSS, CSR
pss=complex(zeros(Npss,numRx));
sss=complex(zeros(Nsss,numRx));
bch=complex(zeros(Nbch,numRx));
pdcch = complex(zeros(numel(idx_ pdcch),numRx));
data=complex(zeros(numel(idx_data),numRx));
idx_dataC=idx_data.';
for n=1:numRx
  grid=in(:,:,n);
  data(:,n)=grid(idx_dataC);                    % Physical Downlink Shared Chan-
nel (PDSCH) = user data
  pdcch(:,n) = grid(idx_ pdcch.');             % Physical Downlink Control Channel (PDCCH)
  if nS==0
    pss(:,n)=grid(idx_PSS.');                   % Primary Synchronization Signal (PSS)
    sss(:,n)=grid(idx_SSS.');                   % Secondary Synchronization Signal (SSS)
    bch(:,n)=grid(idx_bch.');                   % Broadcast Channel data (BCH)
  elseif nS==10
    pss(:,n)=grid(idx_PSS.');                   % Primary Synchronization Signal (PSS)
    sss(:,n)=grid(idx_SSS.');                   % Secondary Synchronization Signal (SSS)
  end
end

%% Cell-specific Reference Signal (CSR) = pilots
switch numTx
  case 1                                        % Case of 1 Tx
    csr=complex(zeros(2*Nrb,4,numRx));          % 4 symbols have CSR  per Subframe
    for n=1:numRx
      grid=in(:,:,n);
      csr(:,:,n)=reshape(grid(idx_csr'), 2*Nrb,4) ;
    end
  case 2                                        % Case of 2 Tx
    idx_0=(1:3:lenOFDM);                        % Total number of Nulls + CSR are constant
    idx_all=[idx_0,  4*lenOFDM+idx_0,  7*lenOFDM+idx_0,  11*lenOFDM+idx_0]';
   csr=complex(zeros(4*Nrb,4,numRx));     % 4 symbols have CSR+NULLs per Subframe
```

```
      for n=1:numRx
         grid=in(:,:,n);
         csr(:, :,n)=reshape(grid(idx_all), 4*Nrb,4) ;
      end
   case 4
      idx_0=(1:3:lenOFDM);                       % Total number of Nulls + CSR are constant
      idx_all=[idx_0,              lenOFDM+idx_0,     4*lenOFDM+idx_0, ...
         7*lenOFDM+idx_0,  8*lenOFDM+idx_0,  11*lenOFDM+idx_0]';
     csr=complex(zeros(4*Nrb,6,numRx));      % 4 symbols have CSR+NULLs  per Subframe
      for n=1:numRx
         grid=in(:,:,n);
         csr(:, :,n)=reshape(grid(idx_all), 4*Nrb,6) ;
      end
end
end
```

6.6.4 基于 CSR 的信道估计

系统线性方程描述的 MIMO 信道可以如下表示：

$$\vec{Y}(n) = \boldsymbol{H}(n) * \vec{X}(n) + \vec{n} \tag{6.1}$$

对时序 n 和给定的子载波，$\vec{Y}(n)$ 为接收的信号，$\vec{X}(n)$ 为发射的信号，$\boldsymbol{H}(n)$ 为信道矩阵，$\vec{n}$ 为 AWGN 向量。当接收端得到接收信号 $\vec{Y}(n)$ 时，我们必须计算信道矩阵 $\boldsymbol{H}(n)$ 和噪声 $\vec{n}$ 的估计以准确估计发射信号 $\vec{X}(n)$。假设信道 AWGN 可估，我们下面将重点讨论估计信道矩阵。

设发射天线数为 numTx，接收天线数为 numRx。信道矩阵的阶数即（numRx，numTx）。对每个子载波和每个 OFDM 符号，必须估计 numRx × numTx 的值。如在前章所讨论的，我们用 CSR（导频）信号进行信道矩阵估计。让我们观察多天线传输如何影响信道估计的过程。考虑例如一个 2 × 2 MIMO 信道，在任意给定时序的 MIMO 系统方程可表示为

$$\begin{bmatrix} y_1(n) \\ y_2(n) \end{bmatrix} = \begin{bmatrix} h_{1,1}(n) h_{1,2}(n) \\ h_{2,1}(n) h_{2,2}(n) \end{bmatrix} * \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} \tag{6.2}$$

注意其中一个接收天线，如 $y_1(n)$，接收信号的值为两个发射天线被两个信道增益调幅后的和：

$$y_1(n) = h_{1,1}(n) * x_1(n) + h_{1,2}(n) * x_2(n) + n_1 \tag{6.3}$$

因多载波传输允许我们通过离散傅里叶变换在频域进行信道估计，我们可以得到信道增益及收发信号间的关系为

$$y_1(\omega) = h_{1,1}(\omega) * x_1(\omega) + h_{1,2}(\omega) * x_2(\omega) + nVar \tag{6.4}$$

式中 $y_1(\omega)$ 为时域信号 $y_1(n)$ 进行傅里叶变换得到，即 $y_1(n) \overset{FFT}{\longleftrightarrow} y_1(\omega)$。nVar 给定子载波 AWGN 信道噪声方差。注意变量 $y_1(\omega)$，$x_1(\omega)$，$x_2(\omega)$ 分别为收发资源网格内给定子载波和给定 OFDM 符号上的接收和发射信号。

假如我们为变量 $x_1(\omega)$ 和 $x_2(\omega)$ 选择已知的导频（CSR）信号，则通过接收信号 $y_1(\omega)$ 并忽略噪声方差我们可以轻松估计信道矩阵变量 $h_{1,1}(\omega)$ 和 $h_{1,2}(\omega)$。在这里频谱零值的需求变得显而易见。在一个给定子载波和给定 OFDM 符号上，表示频谱零值需要 $x_1(\omega)$ 等于导频信号而相同子载波 $x_2(\omega)$ 的值等于零。故上式可以化简为

$$y_1(\omega) = h_{1,1}(\omega) * x_1(\omega)]_{\omega = subcarrier} + h_{1,2}(\omega) * x_2(\omega)]_{\omega = subcarrier}$$

$$y_1(\omega) = h_{1,1}(\omega) * x_1(\omega) + h_{1,2}(\omega) * 0.0$$

$$y_1(\omega) = h_{1,1}(\omega) * x_1(\omega) \tag{6.5}$$

通过资源网格内 CSR 信号和频谱零值，我们可以估计信道矩阵路径增益值 $h_{m,n}(\omega)$ 为

$$h_{m,n}(\omega) = \frac{y_n(\omega)}{x_m(\omega)} \tag{6.6}$$

式中 m 为发射天线序数，$m = 1, \cdots,$ numTx；n 为接收天线序数，$n = 1, \cdots,$ numRx。在下一节中我们会看到在 MATLAB 程序中我们如何用收发端 CSR 信号实现系统方程并估计信道矩阵。随后，通过内插方法在资源网格中扩展信道矩阵，我们可以得到全网格信道频率响应估计。

6.6.5 信道估计函数

下面的 MATLAB 函数表现了如何用收发端参考信号，即分割 OFDM 时 - 频网格的导频信号，进行信道估计。该函数有四个输入：结构体（prmLTE）中得到的 PDSCH 参数、接收 CSR 信号（Rx）、发射 CSR 信号（Ref），和表示信道估计模式（Mode）的参数。函数计算全网格信道频率响应（hD）作为输出。

Algorithm

MATLAB function

```
function hD = ChanEstimate_mTx(prmLTE, Rx, Ref, Mode)
%#codegen
Nrb       = prmLTE.Nrb;    % Number of resource blocks
Nrb_sc    = prmLTE.Nrb_sc;             % 12 for normal mode
Ndl_symb = prmLTE.Ndl_symb;       % 7    for normal mode
numTx     = prmLTE.numTx;
numRx     = prmLTE.numRx;
% Initialize output buffer
switch numTx
```

```
    case 1                                              % Case of 1 Tx
      hD = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2,numRx));  % Initialize Output
      % size(Rx) = [2*Nrb,  4,numRx]  size(Ref) = [2*Nrb, 4]
      Edges=[0,3,0,3];
      for n=1:numRx
        Rec=Rx(:,:,n);
        hp= Rec./Ref;
        hD(:,:,n)=gridResponse(hp, Nrb, Nrb_sc, Ndl_symb, Edges,Mode);
      end
    case 2                                  % Case of 2 Tx
      hD = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2,numTx, numRx));
      % size(Rx) = [4*Nrb,  4,numRx]  size(Ref) = [2*Nrb, 4, numTx]
      for n=1:numRx
        Rec=Rx(:,:,n);
        for m=1:numTx
          [R,Edges]=getBoundaries2(m, Rec);
          T=Ref(:,:,m);
          hp= R./T;
          hD(:,:,m,n)=gridResponse(hp, Nrb, Nrb_sc, Ndl_symb, Edges,Mode);
        end
      end
    case 4
      hD = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2,numTx, numRx));
      % size(Rx) = [4*Nrb,  4,numRx]  size(Ref) = [2*Nrb, 4, numTx]
      for n=1:numRx
        Rec=Rx(:,:,n);
        for m=1:numTx
          [R,idx3, Edges]=getBoundaries4(m, Rec);
          T=Ref(:,idx3,m);
          hp= R./T;
          hD(:,:,m,n)=gridResponse(hp, Nrb, Nrb_sc, Ndl_symb, Edges,Mode);
        end
      end
end
end
%% Helper function
function [R,idx3, Edges]=getBoundaries4(m, Rec)
coder.varsize('Edges');coder.varsize('idx3');
numPN=size(Rec,1);
idx_0=(1:2:numPN);
idx_1=(2:2:numPN);
Edges=[0,3,0,3];
idx3=1:4;
switch m
  case 1
    index=[idx_0, 2*numPN+idx_1, 3*numPN+idx_0, 5*numPN+idx_1]';
    Edges=[0,3,0,3];  idx3=1:4;
```

```
    case 2
      index=[idx_1, 2*numPN+idx_0, 3*numPN+idx_1, 5*numPN+idx_0]';
      Edges=[3,0,3,0];   idx3=1:4;
    case 3
      index=[numPN+idx_0, 4*numPN+idx_1]';
      Edges=[0,3];       idx3=[1 3];
    case 4
      index=[numPN+idx_1, 4*numPN+idx_0]';
      Edges=[3,0];       idx3=[1 3];
end
R=reshape(Rec(index),numPN/2,numel(Edges));
end
%% Helper function
function [R, Edges]=getBoundaries2(m, Rec)
numPN=size(Rec,1);
idx_0=(1:2:numPN);
idx_1=(2:2:numPN);
Edges=[0,3,0,3];
switch m
    case 1
      index=[idx_0, numPN+idx_1, 2*numPN+idx_0, 3*numPN+idx_1]';
      Edges=[0,3,0,3];
    case 2
      index=[idx_1, numPN+idx_0, 2*numPN+idx_1, 3*numPN+idx_0]';
      Edges=[3,0,3,0];
end
R=reshape(Rec(index),numPN/2,4);
end
```

函数进行信道估计的过程分为两步：首先，它由参考信号计算资源网格所有元素构成的信道矩阵。这一步通过遍历所有发射参考信号（T）和接收参考信号（R）以及使用 MATLAB 元素除法器计算信道矩阵元素。然后，调用 gridResponse 函数将信道矩阵估计由通过 CSR 值计算的结果扩展到整个网格。对决定扩展结果的值进行插值或求平均运算，其类型由输入（Mode）决定。下面我们将会看到各种信道扩展操作。

6.6.6 信道估计扩展

下面的 MATLAB 函数包含三个可以在信道频率响应影响全资源网格条件下，单纯用 CSR 信号扩展信道矩阵生成函数输出（y）的算法。函数输入如下：由导频（hp）计算的有限信道频响和与资源网格阶数有关的参数，包括资源块数（Nrb）、每个资源块内子载波数（Nrb _ sc），以及每个时隙 OFDM 数（Ndl _ symb），除此之外还有另外两个：定义 CSR 相对资源块边缘位置（Edges）和全网格扩展频响算法模式（Mode）。

Algorithm

MATLAB function

```
function y=gridResponse(hp, Nrb, Nrb_sc, Ndl_symb, Edges,Mode)
%#codegen
switch Mode
   case 1
      y=gridResponse_interpolate(hp, Nrb, Nrb_sc, Ndl_symb, Edges);
   case 2
      y=gridResponse_averageSlot(hp, Nrb, Nrb_sc, Ndl_symb, Edges);
   case 3
      y=gridResponse_averageSubframe(hp,  Ndl_symb, Edges);
   otherwise
      error('Choose the right Mode in function ChanEstimate.');
end
end
```

下面的 MATLAB 函数（gridResponse _ interpolate. m）设定输入变量 Mode 为 1。这表示扩展算法采用频域和时域内插。该算法在频域包含 CSR 的 OFDM 符号子载波间进行内插。通过计算这些符号全子载波信道响应，函数随后内插求的全资源网格的信道响应。

该算法与用于单天线情况的算法不同之处在于该算法分开处理两天线和四天线配置。注意四天线配置下第 3 和第 4 天线上包含 CSR 信号的 OFDM 符号数为 2。OFDM 符号内插必须考虑到这一细节。

Algorithm

MATLAB function

```
function hD=gridResponse_interpolate(hp, Nrb, Nrb_sc, Ndl_symb, Edges)
% Average over the two same Freq subcarriers, and then interpolate between
% them - get all estimates and then repeat over all columns (symbols).
% The interpolation assmues NCellID = 0.
% Time average two pilots over the slots, then interpolate (F)
% between the 4 averaged values, repeat for all symbols in sframe
Separation=6;
hD = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2));
N=numel(Edges);
% Compute channel response over all resource elements of OFDM symbols
switch N
   case 2
      Symbols=[2, 9];
      % Interpolate between subcarriers
      for n=1:N
         E=Edges(n);Edge=[E, 5-E];
         y = InterpolateCsr(hp(:,n),  Separation, Edge);
```

```
        hD(:,Symbols(n))=y;
      end
      % Interpolate between OFDM symbols
      for m=[1,3:8,10:14]
        alpha=(1/7)*(m-2);
        beta=1-alpha;
        hD(:,m)   = beta*hD(:,2) + alpha*hD(:, 9);
      end
    case 4
      Symbols=[1, 5, 8, 12];
      % Interpolate between subcarriers
      for n=1:N
        E=Edges(n);Edge=[E, 5-E];
        y = InterpolateCsr(hp(:,n), Separation, Edge);
        hD(:,Symbols(n))=y;
      end
      % Interpolate between OFDM symbols
      for m=[2, 3, 4, 6, 7]
        alpha=0.25*(m-1);
        beta=1-alpha;
        hD(:,m)   = beta*hD(:,1) + alpha*hD(:, 5);
        hD(:,m+7) =beta*hD(:,8) + alpha*hD(:,12);
      end
    otherwise
      error('Wrong Edges parameter for function gridResponse.');
end
```

下面的 MATLAB 函数（gridResponse _ averageSlot. m）设定输入变量 Mode 为 2。这表示扩展算法为对时隙上 OFDM 符号在频域上进行内插和平均。该算法判断一个给定时隙上有一个还是两个 OFDM 符号包含 CSR。假如有两个 OFDM 符号包含 CSR 信号，算法从前两个 OFDM 符号中合并 CSR 信号。在此情况下，两个 CSR 信号只间隔 3 个子载波而不是 6 个。假如只有一个 OFDM 包含 CSR 信号（如四天线配置里的第 3 和第 4 天线），CSR 不需要合并。两个 CSR 间隔 6 个子载波。在下一步中，函数根据频轴上 CSR 子载波间隔值进行内插。最后函数将相同的信道响应扩展到一个给定时隙所有的 OFDM 符号并遍历其他所有时隙重复这一过程以计算全资源网格的信道响应。

Algorithm

MATLAB function

```
function hD=gridResponse_averageSlot(hp, Nrb, Nrb_sc, Ndl_symb, Edges)
% Average over the two same Freq subcarriers, and then interpolate between
% them - get all estimates and then repeat over all columns (symbols).
% The interpolation assmues NCellID = 0.
% Time average two pilots over the slots, then interpolate (F)
% between the 4 averaged values, repeat for all symbols in sframe
```

```
Separation=3;
hD = complex(zeros(Nrb*Nrb_sc, Ndl_symb*2));
N=numel(Edges);
% Compute channel response over all resource elements of OFDM symbols
switch N
   case 2
      % Interpolate between subcarriers
      Index=1:Ndl_symb;
      for n=1:N
      E=Edges(n);Edge=[E, 5-E];
      y = InterpolateCsr(hp(:,n),  2* Separation, Edge);
      % Repeat between OFDM symbols in each slot
      yR=y(:,ones(1,Ndl_symb));
      hD(:,Index)=yR;
      Index=Index+Ndl_symb;
      end
   case 4
      Edge=[0 2];
      h1_a_mat = [hp(:,1),hp(:,2)].';
      h1_a = h1_a_mat(:);
      h2_a_mat = [hp(:,3),hp(:,4)].';
      h2_a = h2_a_mat(:);
      hp_a=[h1_a,h2_a];
      Index=1:Ndl_symb;
      for n=1:size(hp_a,2)
      y = InterpolateCsr(hp_a(:,n),  Separation, Edge);
      % Repeat between OFDM symbols in each slot
      yR=y(:,ones(1,Ndl_symb));
      hD(:,Index)=yR;
      Index=Index+Ndl_symb;
      end
   otherwise
      error('Wrong Edges parameter for function gridResponse.');
end
```

下面的程序（gridResponse _ averageSubframe. m）设定输入变量 Mode 为 3。这表示扩展算法对全体子帧 OFDM 符号在频域上内插和平均。算法判断一个给定子帧上包含 CSR 信号的 OFDM 符号是 2 个还是 4 个。假如有 4 个 OFDM 符号包含 CSR，则算法先对第 1 个和第 3 个 OFDM 符号取平均，再对第 2 个和第 4 个取平均，最后合并平均向量。在此情况下，CSR 间隔 3 个子载波而不是 6 个。假如有两个 OFDM 包含 CSR，算法先合并两个 CSR 信号，两个 CSR 信号间隔三个子载波。随后，函数延频轴以 CSR 间隔子载波数 3 进行内插。最后，将相同的信道响应扩展到子帧所有的 OFDM 符号，而得到全资源网格的信道响应。

Algorithm

MATLAB function

```
function hD=gridResponse_averageSubframe(hp, Ndl_symb, Edges)
% Average over the two same Freq subcarriers, and then interpolate between
% them - get all estimates and then repeat over all columns (symbols).
% The interpolation assmues NCellID = 0.
% Time average two pilots over the slots, then interpolate (F)
% between the 4 averaged values, repeat for all symbols in sframe
Separation=3;
N=numel(Edges);
Edge=[0 2];
% Compute channel response over all resource elements of OFDM symbols
switch N
   case 2
      h1_a_mat = hp.';
      h1_a = h1_a_mat(:);
      % Interpolate between subcarriers
      y = InterpolateCsr(h1_a,  Separation, Edge);
      % Repeat between OFDM symbols
      hD=y(:,ones(1,Ndl_symb*2));
   case 4
      h1_a1 = mean([hp(:, 1), hp(:, 3)],2);
      h1_a2 = mean([hp(:, 2), hp(:, 4)],2);
      h1_a_mat = [h1_a1 h1_a2].';
      h1_a = h1_a_mat(:);
      % Interpolate between subcarriers
      y = InterpolateCsr(h1_a,  Separation, Edge);
      % Repeat between OFDM symbols
      hD=y(:,ones(1,Ndl_symb*2));
   otherwise
      error('Wrong Edges parameter for function gridResponse.');
end
```

三个算法体现了不同的动态过程。注意在 OFDM 传输普通循环前缀情况下，每个时隙包含 7 个 OFDM 符号即每个子帧包含 14 个。第一种算法进行信道估计其子帧上的响应是动态且随 OFDM 符号变化的。第二种算法在开始 7 个 OFDM 符号（第一个时隙）得到一个常数信道响应值，而在接下来的 7 个 OFDM 符号（第二个时隙）得到不同的常数信道响应值。第三种算法接近静态，所有子帧的 OFDM 符号拥有一个信道响应值。

6.6.7 理想信道估计

迄今为止，我们讨论了使用导频信号进行信道响应估计的算法。这些算法来

自真实系统并可在现实中实现。在本节中，我们讨论所谓的“理想信道估计器”。这个类型的理想算法根据 MIMO 信道模型提供的确定的信道矩阵或路径增益值进行估计。因 MIMOFadingChan. m 函数的第二个输出为一个反映路径增益的多维矩阵，理想信道估计器可以用这些路径增益计算全资源网格的信道频率响应最优估计。注意因为这种方法是一种理论计算，它无法在现实系统中实现。它只能应用于仿真以得到一个信道估计的边界值。

函数 IdChEst. m 实现了一个理想信道估计器。它的输入为结构体（prmLTEPDSCH）中的 PDSCH 参数、信道模型参数结构体（prmMdl）以及信道矩阵（chPathG）——它为函数 MIMOFadingChan. m 的第二个输出。根据其输出，函数计算全资源网格信道频率响应（H）。

Algorithm

MATLAB function

```
function H = IdChEst(prmLTEPDSCH, prmMdl, chPathG)
% Ideal channel estimation for LTE subframes
%
%   Given the system parameters and the MIMO channel path Gains, provide
%   the ideal channel estimates for the RE corresponding to the data.
%   Limitation - will work for path delays that are multiple of channel sample
%   time and largest pathDelay < size of FFT
%   Implementation based on FFT of channel impulse response
persistent hFFT;
if isempty(hFFT)
  hFFT = dsp.FFT;
end
% get parameters
numDataTones = prmLTEPDSCH.Nrb*12; % Nrb_sc = 12
N                = prmLTEPDSCH.N;
cpLen0           = prmLTEPDSCH.cpLen0;
cpLenR           = prmLTEPDSCH.cpLenR;
Ndl_symb         = prmLTE.Ndl_symb;      % 7   for normal mode
slotLen          = (N*Ndl_symb + cpLen0 + cpLenR*6);
% Get path delays
pathDelays = prmMdl.PathDelays;
% Delays, in terms of number of channel samples, +1 for indexing
sampIdx = round(pathDelays/(1/prmLTEPDSCH.chanSRate)) + 1;
[~, numPaths, numTx, numRx] = size(chPathG);
% Initialize output
H = complex(zeros(numDataTones, 2*Ndl_symb, numTx, numRx));
for i= 1:numTx
   for j = 1:numRx
      link_PathG = chPathG(:, :, i, j);
      % Split this per OFDM symbol
```

```
        g = complex(zeros(2*Ndl_symb, numPaths));
        for n = 1:2 % over two slots
            % First OFDM symbol
            Index=(n-1)*slotLen + (1:(N+cpLen0));
            g((n-1)*Ndl_symb+1, :) = mean(link_PathG(Index, :), 1);
            % Next 6 OFDM symbols
            for k = 1:6
                Index=(n-1)*slotLen+cpLen0+k*N+(k-1)*cpLenR + (1:(N+cpLenR));
                g((n-1)*Ndl_symb+k+1, :) = mean(link_PathG(Index, :), 1);
            end
        end
        hImp = complex(zeros(2*Ndl_symb, N));
        % assign pathGains at impulse response sample locations
        hImp(:, sampIdx) = g;
        % FFT of impulse response
        h = step(hFFT, hImp.');
        % Reorder, remove DC, Unpack channel gains
        h = [h(N/2+1:N, :); h(1:N/2, :)];
        H(:, :, i, j) = [h(N/2-numDataTones/2+1:N/2, :); h(N/2+2:N/2+1+numDataTones/2, :)];
    end
end
```

该函数实质上通过信道冲击响应的快速傅里叶变换计算计算信道频率响应。它对全部子帧取平均值，故一个子帧全部 14 个 OFDM 有相同的信道响应。该函数的处理可总结如下：

1）对任意给定发射天线和接受天线，对所有采样提取信道路径增益；

2）排除循环前缀采样；

3）对所有非循环前缀采样取平均值；

4）初始化单冲击响应向量（hImp）；

5）通过将路径延迟值整数化归一化找到冲击响应的非零采样；

6）通过代入非零采样的路径增益平均值更新冲击响应向量；

7）求冲击响应的 FFT；

8）整理信道响应值并拆分信道增益以得到全资源网格的信道响应。

6.6.8 信道响应提取

接收天线接受资源网格包含多种类型数据，包括用户数据、CSR 和频谱零值信号、DCI、同步信号，和 BCH 信号。为了均衡和恢复用户数据，我们必须从估计的信道响应中提取搭载用户数据的元素。下面的 MATLAB 函数（ExtChResponse. m）利用 PDSCH 参数结构体（prmLTEPDSCH）和用户数据索引（idx _ data）提取与用户数据（hD）有关的网格（chEst）信道响应值。注意当函数调

用时，用户数据索引（idx _ data）为资源反映射函数（REdemapper _ mTx）的第三个输出。

Algorithm

MATLAB function

```
function hD=ExtChResponse(chEst, idx_data, prmLTE)
%#codegen
numTx = prmLTE.numTx;
numRx = prmLTE.numRx;
if (numTx==1)
   hD=complex(zeros(numel(idx_data),numRx));
   for n=1:numRx
      tmp=chEst(:,:,n);
      hD(:,n)=tmp(idx_data);
   end
else
   hD=complex(zeros(numel(idx_data),numTx,numRx));
   for n=1:numRx
      for m=1:numTx
         tmp=chEst(:,:,m,n);
         hD(:,m,n)=tmp(idx_data);
      end
   end
end
```

6.7 MIMO 的特殊特征

在这一节中我们将会介绍 MIMO 实现的一些特殊功能。它们涉及预编码、层映射，和 MIMO 接收器。这些操作随是否使用发射分集和空分复用而不同。通过这些 MIMO 的特殊特征与一般特征相结合——资源网格计算、信道估计，以及 OFDM 信号生成有关的操作——我们可以完整定义 PDSCH 处理。在本章中我们将介绍 LTE 标准中 MIMO 传输模式 2、3，4 的 PDSCH 处理。

6.7.1 发射分集

发射分集在发射端使用多天线提高分集增益和链路质量。LTE 定义了两种发射分集方案：2 ×2 SFBC 技术和 4 ×4 技术。两种技术都提供全速率编码并通过分集提高性能。

6.7.1.1 发射分集中的 MIMO 处理

LTE 标准定义 MIMO 操作为层映射和预编码。在发射分集模式，层映射和预

编码在一个编码内操作完成。发射分集编码器将调制符号分为两个一组并通过分集编码将调制符号对分配到不同的发射天线。因每个发射天线上的采样来自于原始调制符号流，层映可以理解射在后台完成且预编码是一系列共轭和取负的结果。因不同天线的采样实质上对应了相同的调制数据，故发射分集的层为一。

两天线端口

当使用两个发射天线时，LTE 发射分集基于 SFBC。SFBC 是与 STBC 非常接近的技术。使用 STBC 的发射分集应用于 3GPP 和 WiMAX 标准。我们现在对 STBC和 SFBC 技术做一个简短的概述并阐释 SFBC 是如何从 STBC 转换得来的。

STBC 可以认为是一种多天线调制和映射技术，它可实现完全分集和简单的编码译码。STBC 最简单的构成为两天线 Alamouti 编码传输。Alamouti 编码的 STBC，如图 6.5 所示，一组相邻调制符号（s_1，s_2）在第一个采样时间映射到两个天线端口。在随后的采样时间中，符号交换并取共轭（$-s_2^*$，s_1^*）映射到两个天线端口。注意这两个相邻向量是正交的。

SFBC，如图 6.6 所示，两个相邻调制符号（s_1，s_2）直接映射到第一个天线端口的两个相邻采样中。在第二天线端口，同样映射交换共轭符号（$-s_2^*$，s_1^*）。两个天线端口的相邻向量也是正交的。

我们可以通过一个简单的变换从 Alamouti 编码的 STBC 生成 SFBC 输出符号。如图 6.7 所示，我们首先对所有第二个调制符号取共轭并取负，然后再进行 Alamouti 编码与 STBC 映射，即可得到调制输入对的 SFBC 输出。这个过程提高了实现 STBC 和 Alamouti 编码的效率并利于程序代码复用。

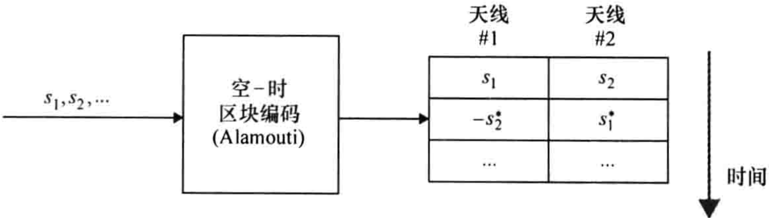


图 6.5 空-时编码：Alamouti 码

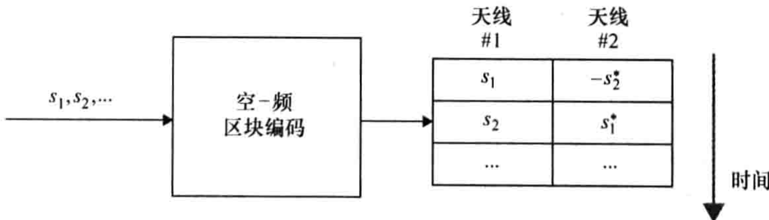


图 6.6 空-频区块编码

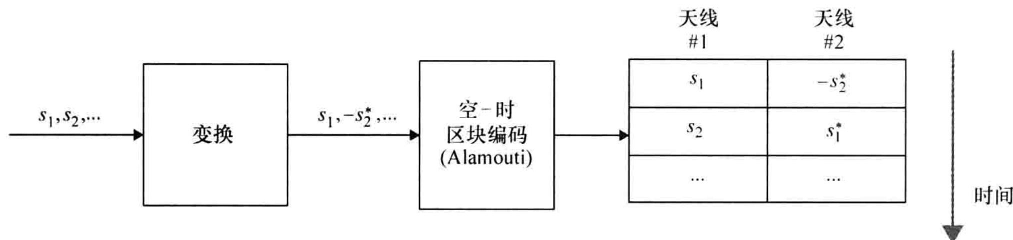


图 6.7　STBC 变换得到 SFBC

四天线端口

当使用四发射天线时，LTE 结合了 SFBC 与频率切换发射分集（FSTD）技术。在这种情况下，我们将四个相邻调制符号进行发射分集编码。首先，我们对第一组调制符号（s_1，s_2）进行 SFBC，并将其输出分配到第一个和第三个天线的前两个采样中。然后我们对第二组调制符号（s_3，s_4）进行 SFBC，并将其输出分配到第二个和第四个天线的第三和第四个采样中。图 6.8 表示了四天线发射分集的处理过程。

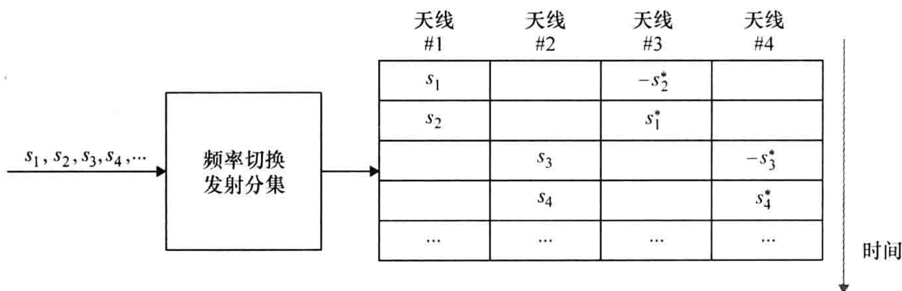


图 6.8　频率切换发射分集结合 SFBC

6.7.1.2　传输分集编码器函数

下面的 MATLAB 函数实现了两种天线配置（两天线和四天线）的传输分集编码器。函数输入为调制符号（in）和发射天线数（numTx）。函数输出（out）为一个 2D 矩阵。输出矩阵的第一阶等于调制符号数，即第一个输入（in）。第二阶的关于发射天线数（numTx）。函数运算过程如下：首先我们将输入中的奇数元素取负并求共轭。假如我们有两个发射天线，则进行 Alamouti 编码 STBC。假如我们有四个发射天线，我们进行 FSTD，从输入中按对采样，每两对进行 alamouti 编码 STBC，并将结果放入输出缓冲区，最后得到求有效值得到输出信号。

Algorithm

MATLAB function

```
function out = TDEncode(in, numTx)
%  Both SFBC and SFBC with FSTD
persistent hTDEnc;
if isempty(hTDEnc)
   % Use same object for either scheme
   hTDEnc = comm.OSTBCEncoder('NumTransmitAntennas', 2);
end
switch numTx
   case 1
      out=in;
   case 2 % SFBC
      in((2:2:end).') = -conj(in((2:2:end).'));
      % STBC Alamouti
      y= step(hTDEnc, in);
      % Scale
      out = y/sqrt(2);
   case 4
      inLen=size(in,1);
      y = complex(zeros(inLen, 4));
      in((2:2:end).') = -conj(in((2:2:end).'));
      idx12 = ([1:4:inLen; 2:4:inLen]); idx12 = idx12(:);
      idx34 = ([3:4:inLen; 4:4:inLen]); idx34 = idx34(:);
      y(idx12, [1 3]) = step(hTDEnc, in(idx12));
      y(idx34, [2 4]) = step(hTDEnc, in(idx34));
      out = y/sqrt(2);
end
```

注意，为了得到 Alamouti 编码 STBC 我们使用通信系统工具箱中的 comm. OSTBCEncoder 系统对象。我们将会在第九章看到，使用系统对象将会提高 STBC 运算效率。

6.7.1.3 发射分集接收器操作

为了得到发射调制符号的最优估计，我们必须在接收端进行发射分集合并。发射分集合并可以理解为是发射分集编码的反过程。

让我们考虑一个 2×2 MIMO 信道。时序（n）时两个接收天线得到的接收信号 $\begin{bmatrix} y_1(n) \\ y_2(n) \end{bmatrix}$，发射信号 $\begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix}$，MIMO 信道矩阵 $\begin{bmatrix} h_{1,1}(n)h_{1,2}(n) \\ h_{2,1}(n)h_{2,2}(n) \end{bmatrix}$，则在时序（$n$）时 MIMO 系统线性方程为

$$\begin{bmatrix} y_1(n) \\ y_2(n) \end{bmatrix} = \begin{bmatrix} h_{1,1}(n)h_{1,2}(n) \\ h_{2,1}(n)h_{2,2}(n) \end{bmatrix} * \begin{bmatrix} x_1(n) \\ x_2(n) \end{bmatrix} \tag{6.7}$$

在下一个时序（$n+1$）时，方程表示为

$$\begin{bmatrix} y_1(n+1) \\ y_2(n+1) \end{bmatrix} = \begin{bmatrix} h_{1,1}(n+1) h_{1,2}(n+1) \\ h_{2,1}(n+1) h_{2,2}(n+1) \end{bmatrix} * \begin{bmatrix} x_1(n+1) \\ x_2(n+1) \end{bmatrix} \tag{6.8}$$

两天线和四天线情况下，发射分集编码对相邻调制符号对进行处理。让我们考虑第一个接收天线上相邻接收采样对 $\begin{bmatrix} y_1(n) \\ y_1(n+1) \end{bmatrix}$ 并代入方程，假设 MIMO 发射端使用了 Alamouti 码 STBC。方程可表示为

$$\begin{bmatrix} y_1(n) \\ y_1(n+1) \end{bmatrix} = \begin{bmatrix} h_{1,1}(n) * x_1(n) + h_{1,2}(n) * x_2(n) \\ h_{1,1}(n+1) * x_1(n+1) + h_{1,2}(n+1) * x_2(n+1) \end{bmatrix} \tag{6.9}$$

该结果可用于任意接收天线端口的任意相邻采样对。

使用 Alamouti 码 STBC 的发射分集分集编码映射调制符号（s_1，s_2）到 2×2 发射信号：

$$\begin{bmatrix} x_1(n) & x_2(n) \\ x_1(n+1) & x_2(n+1) \end{bmatrix} = \begin{bmatrix} s_1 & s_2 \\ -s_2^* & s_1^* \end{bmatrix} \tag{6.10}$$

代入式（6.9），可以得到 2×2 发射分集的接收信号方程：

$$\begin{bmatrix} y_1(n) \\ y_1(n+1) \end{bmatrix} = \begin{bmatrix} h_{1,1}(n) * s_1 + h_{1,2}(n) * s_2 \\ -h_{1,1}(n+1) * s_2^* + h_{1,2}(n+1) * s_1^* \end{bmatrix} \tag{6.11}$$

现在，假设两个相邻采样的信道增益近似相等（即 $h_{1,1}(n) \approx h_{1,1}(n+1) = h_{1,1}$ 和 $h_{1,2}(n) \approx h_{1,2}(n+1) = h_{1,2}$），且确定时序 n 的值（如 $y_1(n) = y_1$ 和 $y_1(n+1) = y_2$），我们可以化简上式为

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} h_{1,1} * s_1 + h_{1,2} * s_2 \\ -h_{1,1} * s_2^* + h_{1,2} * s_1^* \end{bmatrix} \tag{6.12}$$

对方程等好两侧第二行变量取共轭并化简得到：

$$\begin{bmatrix} y_1 \\ y_2^* \end{bmatrix} = \begin{bmatrix} h_{1,1} * s_1 + h_{1,2} * s_2 \\ -h_{1,1}^* * s_2 + h_{1,2}^* * s_1 \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{1,2}^* & -h_{1,1}^* \end{bmatrix} * \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \tag{6.13}$$

通过矩阵 $\boldsymbol{H} = \begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{1,2}^* & -h_{1,1}^* \end{bmatrix}$ 求反，我们可以解出调制发射符号 $\begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix}$ 的最优估计，用接收符号 $\begin{bmatrix} y_1 \\ y_2^* \end{bmatrix}$ 表示，即

$$\begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix} = \begin{bmatrix} h_{1,1} & h_{1,2} \\ h_{1,2}^* & -h_{1,1}^* \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ y_2^* \end{bmatrix}$$

$$\begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix} = \frac{\begin{bmatrix} h_{1,1}^* & h_{1,2} \\ h_{1,2}^* & -h_{1,1} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2^* \end{bmatrix}}{(h_{1,1} * h_{1,1}^* + h_{1,2} * h_{1,2}^*)} \tag{6.14}$$

该方程可以在给定接收天线求解发射符号的估计$\begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix}$。为了计算发射符号的总估计，需要使用 MCR 算法。MCR 算法合并所有接收端估计值，如下文所述。

在每个接收端（标索引 n），设估计 $\vec{s}_n = \begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix}_n$，信道矩阵 $\boldsymbol{H}_n = \begin{bmatrix} h_{1,1}^* & h_{1,2} \\ h_{1,2}^* & -h_{1,1} \end{bmatrix}_n$，接收符号 $\vec{y}_n = \begin{bmatrix} y_1 \\ y_2^* \end{bmatrix}_n$，以及信道矩阵范数（能量估计）

$$E_n = (h_{1,1} * h_{1,1}^* + h_{1,2} * h_{1,2}^*)_n$$

方程（6.14）可以改写为

$$\vec{s}_n = \frac{1}{E_n} H_n \vec{y}_n \tag{6.15}$$

MCR 算法将 N 个接收天线的独立估计值（$\hat{s}$）加权求和得到估计总估计（$\vec{s}_n$），其中 $1 < n < N$，即

$$\hat{s} = \sum_{n=1}^{N} \alpha_n \vec{s}_n \tag{6.16}$$

每个独立估计，在接收端 n，由系数 α_n 加权。

加权系数由给定信道矩阵范数 E_n 与全部信道矩阵范数相除得到，即

$$\alpha_n = \frac{E_n}{\sum_{k=1}^{N} E_k} \tag{6.17}$$

合并式（6.15）~式（6.17）并化简，我们可以得到发射符号最优总估计的最大比合并为

$$\hat{s} = \sum_{n=1}^{N} \alpha_n \vec{s}_n = \sum_{n=1}^{N} \frac{E_n}{\sum_{k=1}^{N} E_k} \cdot \frac{1}{E_n} H_n \vec{y}_n = \frac{\sum_{n=1}^{N} H_n \vec{y}_n}{\sum_{k=1}^{N} E_k} \tag{6.18}$$

下面的 MATLAB 函数实现 2×2 Alamouti 码发射分集合并。函数有两个输入：

1）接收符号（u），阶数为（LEN，2）；

2）估计信道矩阵，阶数为（LEN，2，2）。函数将接收符号分为相邻对并在每个接收天线进行 ML 合并估计。

Algorithm

MATLAB function

```
function s = Alamouti_Combiner1(u,H)
%#codegen
% STBC_DEC STBC Combiner
%   Outputs the recovered symbol vector
LEN=size(u,1);
Nr=size(u,2);
BlkSize=2;
NoBlks=LEN/BlkSize;
% Initialize outputs
h=complex(zeros(1,2));
s=complex(zeros(LEN,1));
% Alamouti code for 2 Tx
indexU=(1:BlkSize);
for m=1:NoBlks
   t_hat=complex(zeros(BlkSize,1));
   h_norm=0.0;
   for n=1:Nr
      h(:)=H(2*m-1,:,n);
      h_norm=h_norm+real(h*h');
      r=u(indexU,n);
      r(2)=conj(r(2));
      shat=[conj(h(1)), h(2); conj(h(2)), -h(1)]*r;
      t_hat=t_hat+shat;
   end
   s(indexU)=t_hat/h_norm; % Maximum-likelihood combining
   indexU=indexU+BlkSize;
end
end
```

这个函数明了地实现了 2 × 2 Alamouti 码 ML 合并的公式。不过，该函数的运行时间并未优化。我们将会在第 9 章中矢量化 MATLAB 函数达到优化运行时间。我们将会使用 comm. OSTBCCombiner 系统对象达到最优性能。下面的 MATLAB 函数使用 comm. OSTBCCombiner 系统对象实现发射分集合并。它只需要 6 行 MATLAB 代码就可完成上面的函数同样的工作。

Algorithm

MATLAB function

```
function s = Alamouti_CombinerS(u,H)
%#codegen
% STBC_DEC STBC Combiner
persistent hTDDec
```

```
if isempty(hTDDec)
   hTDDec= comm.OSTBCCombiner(...
      'NumTransmitAntennas',2,'NumReceiveAntennas',2);
end
s = step(hTDDec, u, H);
```

6.7.1.4 发射分集合并器函数

下面的 MATLAB 函数实现了两天线和四天线配置发射分集合并器。函数输入如下：

1）2D 接收信号（in）；

2）3D 信道估计信号（chEst）；

3）发射天线数（numTx）；

4）接收天线数（numRx）。

函数输出（y）为发射调制信号的 ML 估计。输出向量（y）的样本数等于发射调制符号数（inLen），即输入信号（in 和 chEst）的第一阶。输入信号（in 和 chEst）的第二阶等于发射天线数（numTx）。信道估计信号（chEst）的第三阶等于接收天线数（numRx）。

发射分集合并操作为发射分集编码的反过程。我们首先求输入信号有效值。假如我们有两个发射天线，则进行 STBC 合并。假如有四个发射天线，我们进行 FSTD 合并。首先，我们调整 3D 信道估计矩阵（chEst）为阶数等于（inLen，2，4）的新矩阵（$\boldsymbol{H}$）。然后我们对矩阵 $\boldsymbol{H}$ 进行 STBC 合并，即我们分别重复对发射天线（1，3）和（2，4）进行 Alamouti 编码合并。最后，我们将奇数项元素取负并共轭然后替换原值，将其返回至 SFBC 并计算输出信号。

Algorithm

MATLAB function

```
function y = TDCombine(in, chEst, numTx, numRx)
% LTE transmit diversity combining
%   SFBC and SFBC with FSTD.
inLen = size(in, 1);
Index=(2:2:inLen)';
switch numTx
   case 1
      y=in;
   case 2   % For 2TX - SFBC
      in = sqrt(2) * in; % Scale
      y = Alamouti_CombinerS(in,chEst);
      % ST to SF transformation.
      % Apply blockwise correction for 2nd symbol combining
      y(Index) = -conj(y(Index));
   case 4   % For 4Tx - SFBC with FSTD
```

```
    in = sqrt(2) * in; % Scale
    H = complex(zeros(inLen, 2, numRx));
    idx12 = ([1:4:inLen; 2:4:inLen]); idx12 = idx12(:);
    idx34 = ([3:4:inLen; 4:4:inLen]); idx34 = idx34(:);
    H(idx12, :, :) = chEst(idx12, [1 3], :);
    H(idx34, :, :) = chEst(idx34, [2 4], :);
    y = Alamouti_CombinerS(in, H);
    % ST to SF transformation.
    % Apply blockwise correction for 2nd symbol combining
    y(Index) = -conj(y(Index));
end
```

6.7.2 收发器启动函数

在我们考察各个 MIMO 传输模式模型之前，我们将在本节构建测试、初始化，和可视化函数。这些函数对所有仿真通用并帮助验证每个收发器模型的性能。

6.7.2.1 初始化函数

下面的初始化函数（commlteMIMO _ initialize）设置仿真参数。该函数用于所有 MIMO 模式，包括发射分集和空分复用。第一个输入（txMode）确定使用哪种 MIMO 模式：值为 2 代表发射分集模式，3 为开环空分复用模式，4 为闭环空分复用模式。为了设定 prmLTEPDSCH，prmLTEDLSCH，和 prmMdl 参数结构体，函数分别调用三个函数：prmsPDSCH，prmsPDLSCH，和 prmMdl。

Algorithm

MATLAB function

```
function [prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, ...
   chEstOn, snrdB, maxNumErrs, maxNumBits)
% Create the parameter structures
% PDSCH parameters
CheckAntennaConfig(numTx, numRx);
prmLTEPDSCH = prmsPDSCH(txMode, chanBW, contReg, modType,numTx, numRx);
prmLTEPDSCH.Eqmode=Eqmode;
prmLTEPDSCH.modType=modType;
[SymbolMap, Constellation]=ModulatorDetail(modType);
prmLTEPDSCH.SymbolMap=SymbolMap;
prmLTEPDSCH.Constellation=Constellation;
% DLSCH parameters
prmLTEDLSCH = prmsDLSCH(cRate,maxIter, fullDecode, prmLTEPDSCH);
% Channel parameters
chanSRate   = prmLTEPDSCH.chanSRate;
 prmMdl = prmsMdl(chanSRate,  chanMdl, numTx, numRx, ...
   corrLvl, chEstOn, snrdB, maxNumErrs, maxNumBits);
```

函数 prmsDLSCH 和 prmsMdl 与上一章的函数代码相同。函数 prmLTEPDSCH 针对 MIMO 做出更新。根据传输模式的不同，天线数、信道带宽、使用的调制模式等必要的 PDSCH 处理参数在此函数中进行设置。

Algorithm

MATLAB function

```
function p = prmsPDSCH(txMode, chanBW, contReg, modType, numTx, numRx,
numCodeWords)
%% PDSCH parameters
switch chanBW
   case 1      % 1.4 MHz
      BW = 1.4e6; N = 128; cpLen0 = 10; cpLenR = 9;
      Nrb = 6; chanSRate = 1.92e6;
   case 2      % 3 MHz
      BW = 3e6; N = 256; cpLen0 = 20; cpLenR = 18;
      Nrb = 15; chanSRate = 3.84e6;
   case 3      % 5 MHz
      BW = 5e6; N = 512; cpLen0 = 40; cpLenR = 36;
      Nrb = 25; chanSRate = 7.68e6;
   case 4      % 10 MHz
      BW = 10e6; N = 1024; cpLen0 = 80; cpLenR = 72;
      Nrb = 50; chanSRate = 15.36e6;
   case 5      % 15 MHz
      BW = 15e6; N = 1536; cpLen0 = 120; cpLenR = 108;
      Nrb = 75; chanSRate = 23.04e6;
   case 6      % 20 MHz
      BW = 20e6; N = 2048; cpLen0 = 160; cpLenR = 144;
      Nrb = 100; chanSRate = 30.72e6;
end
p.BW = BW;                 % Channel bandwidth
p.N = N;                   % NFFT
p.cpLen0 = cpLen0;         % Cyclic prefix length for 1st symbol
p.cpLenR = cpLenR;         % Cyclic prefix length for remaining
p.Nrb = Nrb;               % Number of resource blocks
p.chanSRate = chanSRate;   % Channel sampling rate
p.contReg = contReg;
switch txMode
   case 1 % SISO transmission
      p.numTx = numTx;
      p.numRx = numRx;
      numCSRRE_RB = 2*2*2; % CSR, RE per OFDMsym/slot/subframe per RB
      p.numLayers = 1;
      p.numCodeWords = 1;
   case 2 % Transmit diversity
      p.numTx = numTx;
      p.numRx = numRx;
```

```
    switch numTx
      case 1
        numCSRRE_RB = 2*2*2; % CSR, RE per OFDMsym/slot/subframe per RB
      case 2   % 2xnumRx
        % RE - resource element, RB - resource block
        numCSRRE_RB = 4*2*2; % CSR, RE per OFDMsym/slot/subframe per RB
      case 4      % 4xnumRx
        numCSRRE_RB = 4*3*2; % CSR, RE per OFDMsym/slot/subframe per RB
    end
    p.numLayers = 1;
    p.numCodeWords = 1; % for transmit diversity
  case 3 % CDD Spatial multiplexing
    p.numTx = numTx;
    p.numRx = numRx;
    switch numTx
      case 1
        numCSRRE_RB = 2*2*2; % CSR, RE per OFDMsym/slot/subframe per RB
      case 2      % 2x2
        % RE - resource element, RB - resource block
        numCSRRE_RB = 4*2*2; % CSR, RE per OFDMsym/slot/subframe per RB
      case 4      % 4x4
        numCSRRE_RB = 4*3*2; % CSR, RE per OFDMsym/slot/subframe per RB
    end
    p.numLayers = min([p.numTx, p.numRx]);
    p.numCodeWords = 1; % for spatial multiplexing
  case 4 % Spatial multiplexing
    p.numTx = numTx;
    p.numRx = numRx;
    switch numTx
      case 1
        numCSRRE_RB = 2*2*2; % CSR, RE per OFDMsym/slot/subframe per RB
      case 2      % 2x2
        % RE - resource element, RB - resource block
        numCSRRE_RB = 4*2*2; % CSR, RE per OFDMsym/slot/subframe per RB
      case 4      % 4x4
        numCSRRE_RB = 4*3*2; % CSR, RE per OFDMsym/slot/subframe per RB
    end
    p.numLayers = min([p.numTx, p.numRx]);
    p.numCodeWords = numCodeWords; % for spatial multiplexing
end
% For Normal cyclic prefix, FDD mode
p.deltaF = 15e3;   % subcarrier spacing
p.Nrb_sc = 12;     % no. of subcarriers per resource block
p.Ndl_symb = 7;    % no. of OFDM symbols in a slot
%% Modeling a subframe worth of data (=> 2 slots)
numResources = (p.Nrb*p.Nrb_sc)*(p.Ndl_symb*2);
numCSRRE = numCSRRE_RB * p.Nrb;               % CSR, RE per
OFDMsym/slot/subframe per RB
```

```
% Actual PDSCH bits calculation - accounting for PDCCH, PBCH, PSS, SSS
switch p.numTx
  % numRE in control region - minus the CSR
  case 1
    numContRE = (10 + 12*(p.contReg-1))*p.Nrb;
    numBCHRE = 60+72+72+72; % removing the CSR present in 1st symbol
  case 2
    numContRE = (8 + 12*(p.contReg-1))*p.Nrb;
    numBCHRE = 48+72+72+72; % removing the CSR present in 1st symbol
  case 4
    numContRE = (8 + (p.contReg>1)*(8+ 12*(p.contReg-2)))*Nrb;
    numBCHRE = 48+48+72+72; % removing the CSR present in 1,2 symbol
end
numSSSRE=72;
numPSSRE=72;
numDataRE=zeros(3,1);
% Account for BCH, PSS, SSS and PDCCH for subframe 0
numDataRE(1)=numResources-numCSRRE-numContRE-numSSSRE
- numPSSRE-numBCHRE;
% Account for PSS, SSS and PDCCH for subframe 5
numDataRE(2)=numResources-numCSRRE-numContRE-numSSSRE - numPSSRE;
% Account for PDCCH only in all other subframes
numDataRE(3)=numResources-numCSRRE-numContRE;
% Maximum data resources - with no extra overheads (only CSR + data)
p.numResources=numResources;
p.numCSRResources =  numCSRRE;
p.numDataResources = p.numResources - p.numCSRResources;
p.numContRE = numContRE;
p.numBCHRE = numBCHRE;
p.numSSSRE=numSSSRE;
p.numPSSRE=numPSSRE;
p.numDataRE=numDataRE;
% Modulation types , bits per symbol, number of layers per codeword
Qm = 2 * modType;
p.Qm = Qm;
p.numLayPerCW = p.numLayers/p.numCodeWords;
% Maximum data bits - with no extra overheads (only CSR + data)
p.numDataBits = p.numDataResources*Qm*p.numLayPerCW;
numPDSCHBits =numDataRE*Qm*p.numLayPerCW;
p.numPDSCHBits = numPDSCHBits;
p.maxG = max(numPDSCHBits);
```

函数 CheckAntennaConfig 被 commlteMIMO _ initialize 调用。它确保仿真中有效的天线配置。在本书中，我们限定天线配置种类为最大 4 个单天线（1 ×1、1 ×2、1 ×3，1 ×4）、双天线（2 ×2）和四天线（4 ×4）。

Algorithm

MATLAB function

```
function CheckAntennaConfig(numTx, numRx)
MyConfig=[numTx,numRx];
Allowed=[1,1;1,2;1,3;1,4;2,2;4,4];
tmp=MyConfig(ones(size(Allowed,1),1),:);
err=sum(abs(tmp-Allowed),2);
if isempty(find(~err,1))
   Status=0;
else
   Status=1;
end
if ~Status
   disp('Wrong antenna configuration! Allowable configurations are:');
   disp(Allowed);
   error('Please change number of Tx and/or Rx antennas!');
end
```

函数 ModulatorDetail 函数同样被 commlteMIMO _ initialize 调用。根据调制模式的不同，函数使用可视化函数中的星座图和符号映射，并使用 MIMO 接收器函数中的球形解码器（SD）。

Algorithm

MATLAB function

```
function [SymMap, Constellation]=ModulatorDetail(Mode)
%% Initialization
persistent QPSK QAM16 QAM64
if isempty(QPSK)
   QPSK       = comm.PSKModulator(4, 'BitInput', true, ...
     'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
     'CustomSymbolMapping', [0 2 3 1]);
   QAM16      = comm.RectangularQAMModulator(16, 'BitInput',true,...
     'NormalizationMethod','Average power',...
     'SymbolMapping', 'Custom', ...
     'CustomSymbolMapping', [11 10 14 15 9 8 12 13 1 0 4 5 3 2 6 7]);
   QAM64      = comm.RectangularQAMModulator(64, 'BitInput',true,...
     'NormalizationMethod','Average power',...
     'SymbolMapping', 'Custom', ...
     'CustomSymbolMapping', [47 46 42 43 59 58 62 63 45 44 40 41 ...
     57 56 60 61 37 36 32 33 49 48 52 53 39 38 34 35 51 50 54 55 7 ...
     6 2 3 19 18 22 23 5 4 0 1 17 16 20 21 13 12 8 9 25 24 28 29 15 ...
     14 10 11 27 26 30 31]);
end
```

```
%% Processing
switch Mode
  case 1
    Constellation=constellation(QPSK);
    SymMap = QPSK.CustomSymbolMapping;
  case 2
    Constellation=constellation(QAM16);
    SymMap = QAM16.CustomSymbolMapping;
  case 3
    Constellation=constellation(QAM64);
    SymMap = QAM64.CustomSymbolMapping;
  otherwise
    error('Invalid Modulation Mode. Use {1,2, or 3}');
end
```

6.7.2.2 可视化函数

在本节中我们更新 zVisualize 函数，以使我们可以直接观察 MIMO 接收器处理前后衰落效应对发射符号的影响。

Algorithm

MATLAB function

```
function zVisualize(prmLTE, txSig, rxSig, yRec, dataRx, csr, nS)
% Constellation Scopes & Spectral Analyzers
zVisConstell(prmLTE, yRec, dataRx, nS);
zVisSpectrum(prmLTE, txSig, rxSig, yRec, csr, nS);
```

函数执行两个进程。首先，调用 zVisConstell 函数显示用户数据在均衡前后的星座图。根据发射天线数不同，函数由通信系统工具箱生成和配置多星座图系统对象。

Algorithm

MATLAB function

```
function zVisConstell(prmLTE, yRec, dataRx, nS)
% Constellation Scopes
switch prmLTE.numTx
  case 1
    zVisConstell_1(prmLTE, yRec, dataRx, nS);
  case 2
    zVisConstell_2(prmLTE, yRec, dataRx, nS);
  case 4
    zVisConstell_4(prmLTE, yRec, dataRx, nS);
end
end
%% Case of numTx =1
```

```
function zVisConstell_1(prmLTE, yRec, dataRx, nS)
persistent h1 h2
if isempty(h1)
   h1 = comm.ConstellationDiagram('SymbolsToDisplay',...
      prmLTE.numDataResources, 'ReferenceConstellation', prmLTE.
Constellation,...
      'YLimits', [-2 2], 'XLimits', [-2 2], 'Position', ...
      figposition([5 60 20 25]), 'Name', 'Before Equalizer');
   h2 = comm.ConstellationDiagram('SymbolsToDisplay',...
      prmLTE.numDataResources, 'ReferenceConstellation', prmLTE.
Constellation,...
      'YLimits', [-2 2], 'XLimits', [-2 2], 'Position', ...
      figposition([6 61 20 25]), 'Name', 'After Equalizer');
end
% Update Constellation Scope
if (nS~=0 && nS~=10)
   step(h1, dataRx(:,1));
   step(h2, yRec(:,1));
end
end
%% Case of numTx =2
function zVisConstell_2(prmLTE, yRec, dataRx, nS)
persistent h11 h21 h12 h22
if isempty(h11)
   h11 = comm.ConstellationDiagram('SymbolsToDisplay',...
      prmLTE.numDataResources, 'ReferenceConstellation', prmLTE.Constellation,...
      'YLimits', [-2 2], 'XLimits', [-2 2], 'Position', ...
      figposition([5 60 20 25]), 'Name', 'Before Equalizer');
   h21 = comm.ConstellationDiagram('SymbolsToDisplay',...
      prmLTE.numDataResources, 'ReferenceConstellation', prmLTE.Constellation,...
      'YLimits', [-2 2], 'XLimits', [-2 2], 'Position', ...
      figposition([6 61 20 25]), 'Name', 'After Equalizer');
   h12 = clone(h11);
   h22 = clone(h21);
end
yRecM = sqrt(2) *TDEncode( yRec, 2);
% Update Constellation Scope
if (nS~=0 && nS~=10)
   step(h11, dataRx(:,1));
   step(h21, yRecM(:,1));
   step(h12, dataRx(:,2));
   step(h22, yRecM(:,2));
end
end
%% Case of numTx =4
function zVisConstell_4(prmLTE, yRec, dataRx, nS)
persistent ha1 hb1 ha2 hb2 ha3 hb3 ha4 hb4
if isempty(ha1)
```

```
    ha1 = comm.ConstellationDiagram('SymbolsToDisplay',...
      prmLTE.numDataResources, 'ReferenceConstellation', prmLTE.Constellation,...
      'YLimits', [-2 2], 'XLimits', [-2 2], 'Position', ...
      figposition([5 60 20 25]), 'Name', 'Before Equalizer');
    hb1 = comm.ConstellationDiagram('SymbolsToDisplay',...
      prmLTE.numDataResources, 'ReferenceConstellation', prmLTE.Constellation,...
      'YLimits', [-2 2], 'XLimits', [-2 2], 'Position', ...
      figposition([6 61 20 25]), 'Name', 'After Equalizer');
    ha2 = clone(ha1);
    hb2 = clone(hb1);
    ha3 = clone(ha1);
    hb3 = clone(hb1);
    ha4 = clone(ha1);
    hb4 = clone(hb1);
end
yRecM = sqrt(2) *TDEncode( yRec, 4);
% Update Constellation Scope
if (nS~=0 && nS~=10)
    step(ha1, dataRx(:,1));
    step(hb1, yRecM(:,1));
    step(ha2, dataRx(:,2));
    step(hb2, yRecM(:,2));
    step(ha3, dataRx(:,3));
    step(hb3, yRecM(:,3));
    step(ha4, dataRx(:,4));
    step(hb4, yRecM(:,4));
end
end
```

然后，zVisualize 函数通过调用 zVisSpectrum 函数显示发射信号的频谱和接收信号均衡前后的频谱。根据发射天线数不同，函数由 DSP 系统工具箱生成和配置多频谱分析器系统对象。

Algorithm

MATLAB function

```
function zVisSpectrum(prmLTE, txSig, rxSig, yRec, csr, nS)
% Spectral Analyzers
switch prmLTE.numTx
    case 1
        zVisSpectrum_1(prmLTE, txSig, rxSig, yRec, csr, nS);
    case 2
        zVisSpectrum_2(prmLTE, txSig, rxSig, yRec, csr, nS);
    case 4
        zVisSpectrum_4(prmLTE, txSig, rxSig, yRec, csr, nS);
end
end
```

```
%% Case of numTx = 1
function zVisSpectrum_1(prmLTE, txSig, rxSig, yRec, csr, nS)
persistent hSpecAnalyzer
if isempty(hSpecAnalyzer)
   hSpecAnalyzer = dsp.SpectrumAnalyzer('SampleRate', prmLTE.chanSRate, ...
      'SpectrumType', 'Power density', 'PowerUnits', 'dBW', ...
      'RBWSource', 'Property', 'RBW', 15000,...
      'FrequencySpan', 'Span and center frequency',...
      'Span', prmLTE.BW, 'CenterFrequency', 0,...
      'FFTLengthSource', 'Property', 'FFTLength', prmLTE.N,...
      'Title', 'Transmitted & Received Signal Spectrum', 'YLimits', [-110 -60],...
      'YLabel', 'PSD');
end
alamoutiRx = TDEncode(yRec, prmLTE.numTx);
yRecGrid = REmapper_mTx(alamoutiRx, csr, nS, prmLTE);
yRecGridSig = lteOFDMTx(yRecGrid, prmLTE);
step(hSpecAnalyzer, ...
   [SymbSpec(txSig(:,1), prmLTE), SymbSpec(rxSig(:,1), prmLTE),
SymbSpec(yRecGridSig(:,1), prmLTE)]);
end
%% Case of numTx = 2
function zVisSpectrum_2(prmLTE, txSig, rxSig, yRec, csr, nS)
persistent hSpec1 hSpec2
if isempty(hSpec1)
   hSpec1 = dsp.SpectrumAnalyzer('SampleRate', prmLTE.chanSRate, ...
      'SpectrumType', 'Power density', 'PowerUnits', 'dBW', ...
      'RBWSource', 'Property', 'RBW', 15000,...
      'FrequencySpan', 'Span and center frequency',...
      'Span', prmLTE.BW, 'CenterFrequency', 0,...
      'FFTLengthSource', 'Property', 'FFTLength', prmLTE.N,...
      'Title', 'Transmitted & Received Signal Spectrum', 'YLimits', [-110 -60],...
      'YLabel', 'PSD');
   hSpec2 = clone(hSpec1);
end
alamoutiRx = TDEncode(yRec, prmLTE.numTx);
yRecGrid = REmapper_mTx(alamoutiRx, csr, nS, prmLTE);
yRecGridSig = lteOFDMTx(yRecGrid, prmLTE);
step(hSpec1, ...
   [SymbSpec(txSig(:,1), prmLTE), SymbSpec(rxSig(:,1), prmLTE),
SymbSpec(yRecGridSig(:,1), prmLTE)]);
step(hSpec2, ...
   [SymbSpec(txSig(:,2), prmLTE), SymbSpec(rxSig(:,2), prmLTE),
SymbSpec(yRecGridSig(:,2), prmLTE)]);
end
%% Case of numTx = 4
function zVisSpectrum_4(prmLTE, txSig, rxSig, yRec, csr, nS)
persistent hSpec1 hSpec2 hSpec3 hSpec4
if isempty(hSpec1)
```

```
  hSpec1 = dsp.SpectrumAnalyzer('SampleRate', prmLTE.chanSRate, ...
    'SpectrumType', 'Power density', 'PowerUnits', 'dBW', ...
    'RBWSource', 'Property', 'RBW', 15000,...
    'FrequencySpan', 'Span and center frequency',...
    'Span', prmLTE.BW, 'CenterFrequency', 0,...
    'FFTLengthSource', 'Property', 'FFTLength', prmLTE.N,...
    'Title', 'Transmitted & Received Signal Spectrum', 'YLimits', [-110 -60],...
    'YLabel', 'PSD');
  hSpec2 = clone(hSpec1);
  hSpec3 = clone(hSpec1);
  hSpec4 = clone(hSpec1);
end
alamoutiRx = TDEncode(yRec, prmLTE.numTx);
yRecGrid = REmapper_mTx(alamoutiRx, csr, nS, prmLTE);
yRecGridSig = lteOFDMTx(yRecGrid, prmLTE);
step(hSpec1, ...
  [SymbSpec(txSig(:,1), prmLTE), SymbSpec(rxSig(:,1), prmLTE),
SymbSpec(yRecGridSig(:,1), prmLTE)]);
step(hSpec2, ...
  [SymbSpec(txSig(:,2), prmLTE), SymbSpec(rxSig(:,2), prmLTE),
SymbSpec(yRecGridSig(:,2), prmLTE)]);
step(hSpec3, ...
  [SymbSpec(txSig(:,3), prmLTE), SymbSpec(rxSig(:,3), prmLTE),
SymbSpec(yRecGridSig(:,3), prmLTE)]);
step(hSpec4, ...
  [SymbSpec(txSig(:,4), prmLTE), SymbSpec(rxSig(:,4), prmLTE),
SymbSpec(yRecGridSig(:,4), prmLTE)]);
end
%% Helper function
function y = SymbSpec(in, prmLTE)
N = prmLTE.N;
cpLenR = prmLTE.cpLen0;
y = complex(zeros(N+cpLenR, 1));
% Use the first Tx/Rx antenna of the input for the display
y(:,1) = in(end-(N+cpLenR)+1:end, 1);
end
```

6.7.3 下行链路传输模式 2

下面的 MATLAB 函数表示了发射分集模式 2 的收发器模型。它包括双天线和四天线配置。LTE 定义的 2×2 和 4×4 方案都是全码率编码，且与单天线比较有较大性能提升。本例中的关键组件包括：

1）单子帧载荷数据（传输块）生成；

2）DLSCH 处理：传输块添加 CRC、码块分段、1/3 码率 Turbo 编码、码率匹配，以及码块级联以生成 PDSCH 码字输入；

3）PDSCH 发射端处理：比特级绕码、数据调制、层映射和二天线或四天线配置的发射分集预编码、以及资源元素映射和 OFDM 信号生成；

4）信道建模：MIMO 衰落信道附加 AWGN 信道；

5）PDSCH 接收端处理：OFDM 信号接收端生成资源网格、资源元素反映射分割 CSR 信号、信道估计、信用信道估计进行 SFBC 合并并软判决译码和去绕码、DLSCH 译码。

Algorithm

MATLAB function

```
function [dataIn, dataOut, modOut, rxSig, dataRx, yRec, csr_ref]...
   = commlteMIMO_TD_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl)
%% TX
%  Generate payload
dataIn = genPayload(nS,  prmLTEDLSCH.TBLenVec);
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
[data, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH, prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
% TD with SFBC
numTx=prmLTEPDSCH.numTx;
alamouti = TDEncode(modOut(:,1),numTx);
% Generate Cell-Specific Reference (CSR) signals
csr = CSRgenerator(nS, numTx);
csr_ref=complex(zeros(2*prmLTEPDSCH.Nrb, 4, numTx));
for m=1:numTx
   csr_pre=csr(1:2*prmLTEPDSCH.Nrb,:,:,m);
   csr_ref(:,:,m)=reshape(csr_pre,2*prmLTEPDSCH.Nrb,4);
end
% Resource grid filling
txGrid = REmapper_mTx(alamouti, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
%% Channel : MIMO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
% Add AWG noise
nVar = 10.^(0.1.*(-snrdB));
rxSig =  AWGNChannel(rxFade, nVar);
%% RX
% OFDM Rx
```

```
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = REdemapper_mTx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
   chEst = ChanEstimate_mTx(prmLTEPDSCH, csrRx,  csr_ref, prmMdl.chEstOn);
   hD     = ExtChResponse(chEst, idx_data, prmLTEPDSCH);
else
   idealChEst = IdChEst(prmLTEPDSCH, prmMdl, chPathG);
   hD =  ExtChResponse(idealChEst, idx_data, prmLTEPDSCH);
end
% Frequency-domain equalizer
if (numTx==1)
   % Based on Maximum-Combining Ratio (MCR)
   yRec = Equalizer_simo(dataRx, hD, nVar, prmLTEPDSCH.Eqmode);
else
   % Based on Transmit Diversity  with SFBC combiner
   yRec = TDCombine(dataRx, hD, prmLTEPDSCH.numTx, prmLTEPDSCH.numRx);
end
% Demodulate
demodOut = DemodulatorSoft(yRec, prmLTEPDSCH.modType, nVar);
% Descramble received codeword
rxCW =  lteDescramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes - CB segmentation, turbo decoding, rate dematching
[decTbData1, ~,~] = lteTbChannelDecoding(nS, rxCW, Kplus1, C1,  prmLTEDLSCH,
prmLTEPDSCH);
% Transport block CRC detection
[dataOut, ~] = CRCdetector(decTbData1);
end
```

6.7.3.1 收发器模型的结构

下面的 MATLAB 脚本调用 MIMO 收发器函数 commlteMIMO。首先，调用初始化函数（commlteMIMO _ initialize）以示这相关参数结构（pdmLTEDLSCH，prmLTEPDSCH，prmMdl）。然后用一个 While 循环调用 MIMO 收发器函数 commlteMIMO _ TD _ step 执行子帧处理。最后，计算 BER 并调用可视化函数显示均衡前后的信道响应和调制星座图。

Algorithm

MATLAB function

```
% Script for MIMO LTE (mode 2)
%
% Single codeword transmission only,
%
clear all
```

```
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, chEstOn, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl corrLvl chEstOn snrdB maxNumErrs maxNumBits
%%
disp('Simulating the LTE Mode 2: Multiple Tx & Rx antennas with transmit diversity');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
hPBer = comm.ErrorRate;
snrdB=prmMdl.snrdB;
maxNumErrs=prmMdl.maxNumErrs;
maxNumBits=prmMdl.maxNumBits;
%% Simulation loop
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
  [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
     commlteMIMO_TD_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
   % Calculate  bit errors
   Measures = step(hPBer, dataIn, dataOut);
    % Visualize constellations and spectrum
   if visualsOn, zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);end;
   % Update subframe number
   nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
end
disp(Measures);
```

6.7.3.2 验证收发器性能

通过执行 MIMO 模型 MATLAB 脚本（commlteMIMO），我们可以通过观察各种信号评估系统性能。仿真使用的参数总结在下面的 MATLAB 脚本中（commlteMIMO _ params）。这些参数定义了发射分集 MIMO 模式的收发器模型，收发天线数为二，信道带宽为 10MHz（每个子帧有一个 OFDM 符号携带 DCI），16QAM（正交幅度调制）调制类型（有早期终止机制的 1/3 码率 Turbo 编译码，最大迭代次数为 6），以及一个多普勒频移为 70MHz 的频率选择性 MIMO 信道（信道估计根据内插算法并使用和 MIMO 接收器一样的发射分集合并器）。在这个仿真中，程序处理 1000 万比特，AWGN 信道的 SNR 设置为 16dB，可进行可视化。

Algorithm

MATLAB function

```
% PDSCH
txMode      = 2;  % Transmission mode one of {1, 2, 4}
numTx       = 2;   % Number of transmit antennas
numRx       = 2;   % Number of receive antennas
chanBW      = 4;  % Index to chanel bandwidth used [1,....6]
contReg     = 1;  % No. of OFDM symbols dedictaed to control information [1,...,3]
modType     = 2;  % Modulation type [1, 2, 3] for ['QPSK,'16QAM','64QAM']
% DLSCH
cRate       = 1/3; % Rate matching target coding rate
maxIter     = 6;   % Maximum number of turbo decoding terations
fullDecode  = 0;   % Whether "full" or "early stopping" turbo decoding is performed
% Channel model
chanMdl     = 'frequency-selective-high-mobility';
corrLvl     = 'Medium';
% Simulation parameters
Eqmode      = 2;     % Type of equalizer used [1,2] for ['ZF', 'MMSE']
chEstOn     = 1;     % One of [0,1,2,3] for 'Ideal estimator','Interpolation',
Slot average','Subframe average'
snrdB       = 16;  % Signal to Noise ratio
maxNumErrs = 5e5; % Maximum number of errors found before simulation stops
maxNumBits = 5e5;  % Maximum number of bits processed before simulation stops
visualsOn   = 1;     % Whether to visualize channel response and constellations
```

图 6.9 所示为每个子帧两个接收天线上用户数据均衡前（第一行）后（第二行）的星座图。我们可以看到均衡器可以补偿信道衰落，在星座图上使补偿后的信号更接近 16QAM。

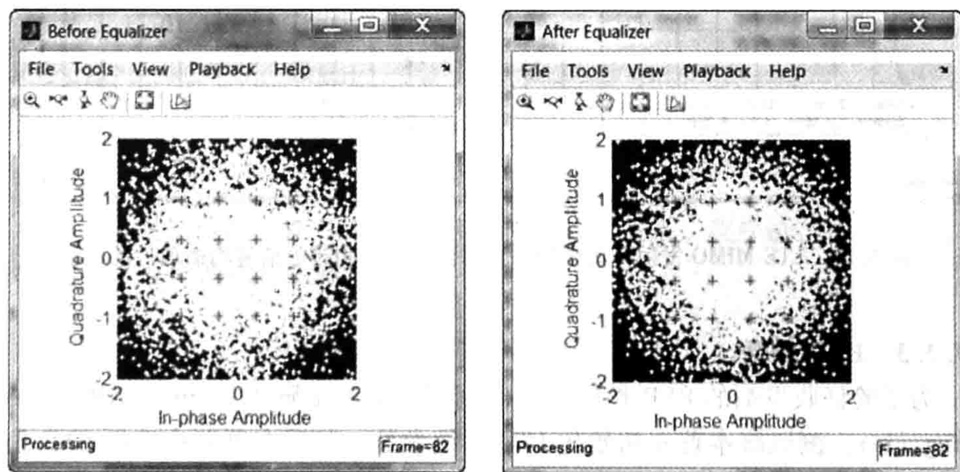


图 6.9 LTE 模型：用户数据在均衡前后的 MIMO 发射分集星座图

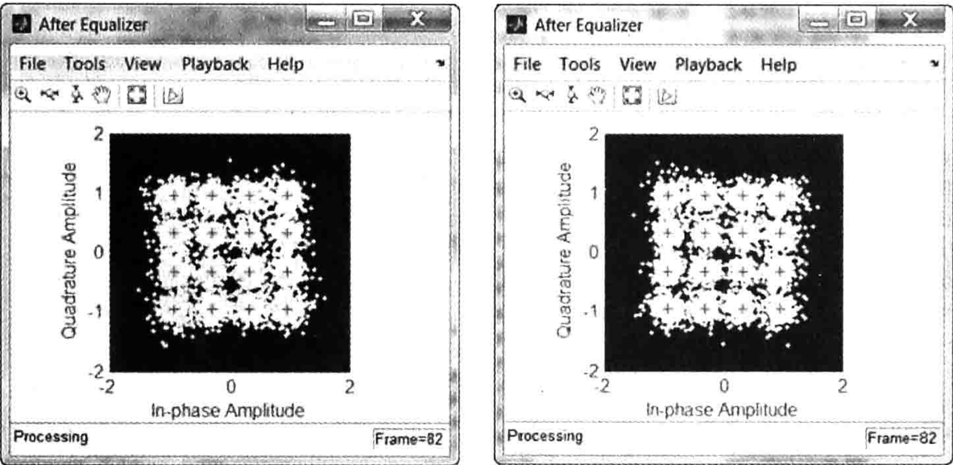


图 6.9　LTE 模型：用户数据在均衡前后的 MIMO 发射分集星座图（续）

图 6.10 所示为每个子帧两个接收天线上用户数据均衡前后的频谱。均衡前的接收信号（可以看到频率选择性衰落的影响）由发射分集均衡后更接近于发射信号的频谱。

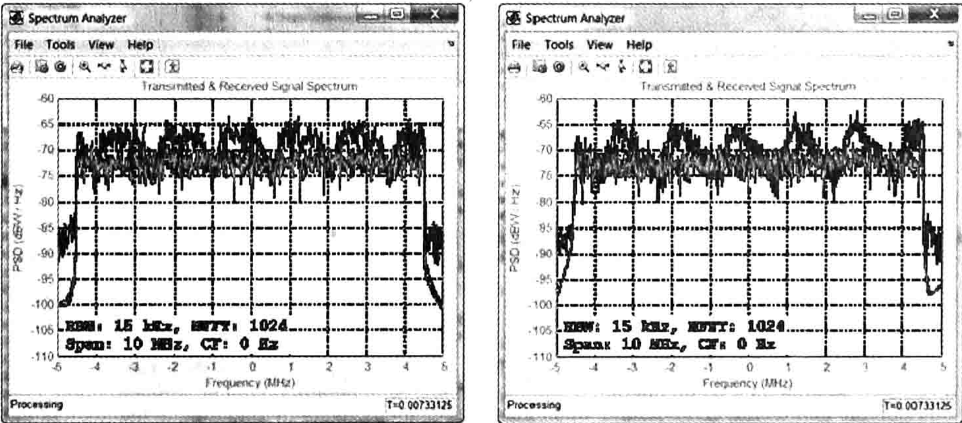


图 6.10　LTE MIMO 发射分集模型：发射信号频谱和接收信号均衡前后的频谱

6.7.3.3　BER 测量

为了验证收发端的 BER 性能，我们创建一个测试脚本 commlteMIMO _ test _ timing _ ber。测试脚本首先初始化 LTE 系统参数，随后在循环中遍历 SNR 值并调用 commlteMIMO _ fcn 函数计算相应的 BER 值。

Algorithm

MATLAB script: commlteMIMO_test_timing_ber

```
% Script for MIMO LTE (mode 2)
%
% Single codeword transmission only,
%
clear all
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_params;
maxNumErrs=5e7;
maxNumBits=5e7;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, chEstOn, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl corrLvl chEstOn snrdB maxNumErrs maxNumBits
%%
disp('Simulating the LTE Mode 2: Multiple Tx & Rx antrennas with transmit diversity');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
%%
MaxIter=8;
snr_vector=getSnrVector(prmLTEPDSCH.modType, MaxIter);
ber_vector=zeros(size(snr_vector));
tic;
for n=1:MaxIter
   fprintf(1,'Iteration %2d out of %2d:  Processing %10d bits. SNR = %3d\n', ...
      n, MaxIter, prmMdl.maxNumBits, snr_vector(n));
   [ber, ~] = commlteMIMO_fcn(snr_vector(n), prmLTEPDSCH, prmLTEDLSCH, prmMdl);
   ber_vector(n)=ber;
end;
toc;
semilogy(snr_vector, ber_vector);
title('BER - commlteMIMO TD');xlabel('SNR (dB)');ylabel('ber');grid;
```

图6.11所示为收发器BER随SNR不同的变化关系。程序进行八次迭代，每次处理5000万比特用户数据。

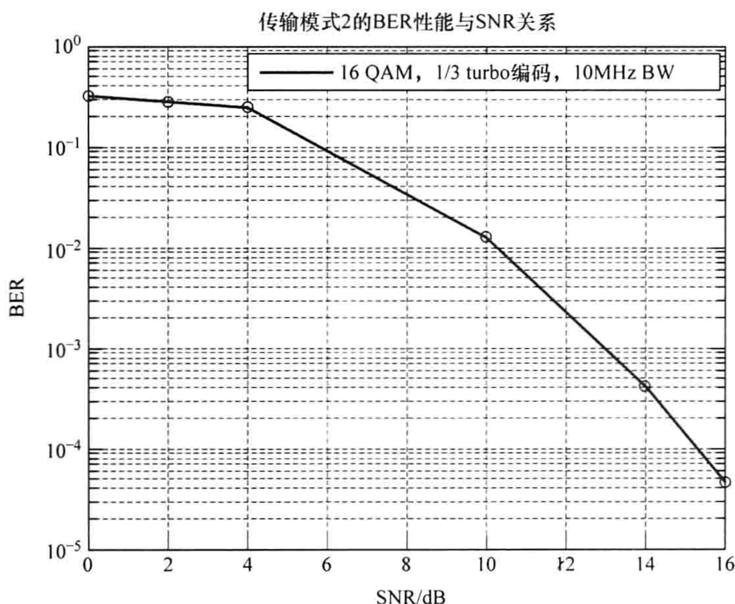


图 6.11　BER 结果：LTE 模式 2，发射分集，2×2 MIMO 信道

6.7.4　空分复用

空分复用技术通过差分比特流使多天线 MIMO 无线系统获得高频谱效率。因为拆分后的子比特流独立调制，空分复用可以比同等空－时或空－频区块码得到更高数据速率。不过，由于发射信号往往缺乏冗余性，致使空分复用容易出现 MIMO 方程矩阵秩不足的问题。计算 MIMO 矩阵的信道估计误差会严重限制性能提升。因此，LTE 标准引入多种机制，包括采用基于秩估计的自适应预编码和层映射，改善在信道劣化情况下的稳健性。

在本节中，我们将详细讨论 LTE 标准中 MIMO 传输采用的空分复用。它包括实现在 OFDM 信号生成并通过多天线传输的同时进行预编码和层映射的方法。然后，通过考量接收端处理，包括各种 MIMO 均衡算法，我们会研究各种条件下的系统性能。

6.7.4.1　预编码技术的原动力

MIMO 在频谱效率方面的长处体现在高散射环境条件下。一个 MIMO 信道在高散射情况下会在每个发射天线和接收天线间形成独立多径链路。因此，收发天线对间的信道增益矩阵满秩，故 MIMO 方程可解。

在一个典型的 MIMO 传输中，有关高散射的假设有时很难保证。因此，为了设计一个实用系统，我们在设计上必须尽可能避免信道矩阵出现秩不足的情况。

LTE 采用的预编码即这样一种最大限度解决秩不足问题的方法。在本节中，我们会考察信道矩阵的秩不足问题，介绍预编码构想，对预编码进行波束赋形内插，并介绍 LTE 标准中不同的预编码种类。随后我们会用 MATLAB 程序实现预编码操作。

6.7.4.2 秩不足问题

空分复用求解下面的系统线性方程，发射信号（X）通过 MIMO 信道矩阵（$\boldsymbol{H}$）并收噪声（n）影响得到接收信号（Y）。方程可表示为

$$Y = \boldsymbol{H}X + n \tag{6.19}$$

例如，一个 4 ×4 MIMO 配置的接收向量 $\vec{Y}$ 可以表示为

$$\vec{Y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,4} \\ \vdots & \ddots & \vdots \\ h_{4,1} & \cdots & h_{4,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{bmatrix} \tag{6.20}$$

当连接收发天线间的路径相似，则矩阵的有些行或列会呈线性相关；例如，下面的这个矩阵的第一和第二行：

$$\boldsymbol{H} = \begin{bmatrix} h_{1,1}h_{1,2} & h_{1,3}h_{1,4} \\ h_{1,1}h_{1,2} & h_{1,3}h_{1,4} \\ h_{3,1}h_{3,2} & h_{3,3}h_{3,4} \\ h_{4,1}h_{4,2} & h_{4,3}h_{4,4} \end{bmatrix} \tag{6.21}$$

在这种情况下，信道矩阵的秩（非线性相关数）为三，而矩阵的阶数为四。这样的系统线性方程为奇异矩阵且不可求反。因此，该 MIMO 系统方程由于矩阵线性相关而不可解。

6.7.4.3 预编码理论

预编码技术可以解决秩不足的问题。一个最优的预编码器由奇异值分解信道矩阵确定。奇异值分解信道矩阵可表示为

$$\boldsymbol{H} = \boldsymbol{UDV} \tag{6.22}$$

式中 $\boldsymbol{V}$ 为阶数等于信道矩阵的秩的正方矩阵；$\boldsymbol{D}$ 为对焦元素为信道矩阵奇异值的对焦矩阵，$\boldsymbol{U}$ 为阶数等于接收天线数的正方矩阵。参考文献［4］和［5］，一个理论上的最优预编码器可以定义为矩阵 $\boldsymbol{V}$ 的列置换矩阵。这个预编码器要求发射端天线有足够的秩以保证 MIMO 方程可解。

像这样的最优编码器无法在现实中实现，因其需要在发射端对信道矩阵可知。而信道矩阵只能在接收端估计，将估计的结果反馈到发射端需要额外占据相当部分的带宽。LTE 选择更有实现性的方法实现预编码。这个方法基于预判决预编码矩阵的有限集。通过这一与向量量化相似的处理，我们可以在收发端得到最

优预编码器。

在发射端，预编码为一个矩阵与层映射之后的调制符号相乘。预编码 MIMO 可表示为

$$Y = \boldsymbol{HV}X + n \tag{6.23}$$

式中　$\boldsymbol{V}$ 为预编码矩阵。在接收端，MIMO 接收器操作之后，我们可将接收信号与发射端预编码信号 $\boldsymbol{V}$ 的逆矩阵相乘。LTE 定义预编码矩阵为 Hermitian 矩阵，即预编码器矩阵为由正交化向量组成。这标志着预编码器的逆矩阵为单纯的 Hermitian 转置矩阵。因为转置一个矩阵的计算复杂性小于求逆矩阵，故预编码可以更高效的执行。

6.7.4.4　预编码矩阵码书

LTE 中的预编码器矩阵的有限集即预编码器码书。表 6.3 表示了两发射天线的预编码器码书。

预编码操作实质上是通过克服秩不足的问题，分散输入信号和减少误码率。通过将预编码器矩阵的列作为波束赋形向量，可以解释预编码较少秩不足发生概率。对单层传输的情况，比如，任意给定码书索引对应一个发射信号 X 和不同波束赋形向量的乘积。而该乘积的几何意义在于它将发射信号旋转到不同的方向。因预编码向量为正交的，故这些不同的方向组成的集合即 $\left\{0, \pi, \frac{\pi}{2}, -\frac{\pi}{2}\right\}$。相位差越大标志着不同比特流通过的路径越不相似。而路径越不相似，则信道矩阵相关性行或列出现的概率越小，即秩不足出现的概率越小。

表 6.3　LTE 空分复用两发射天线配置下的预编码矩阵

码书索引	层数	
	1	2
0	$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}$	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$
1	$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$
2	$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\j\end{bmatrix}$	$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\j & -j\end{bmatrix}$
3	$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\-j\end{bmatrix}$	—

6.7.4.5　预编码种类

开环和闭环 MIMO 中都可以使用预编码。开环预编码用于 MIMO 传输模式

3，而闭环预编码用于传输模式 4。在快换预编码中，收发端用预编码矩阵索引的预定义集，并根据这个预定义集周期性旋转，而不需要传输任何码书。闭环预编码反馈驱动接收端从有限码书选择预编码矩阵，并将所选矩阵用有限个比特传输到发射端。预编码矩阵码书选择和闭环预编码矩阵反馈将在第 7 章中讨论。

6.7.5 空分复用中的 MIMO 操作

LTE 标准中的九种传输模式中，有六种基于空分复用。空分复用的显著特点在于层映射和预编码。因为每个发射天线的采样都独立与其他天线，正交调制流映射到不同的子调制流并被发射天线传送。因不同的天线有不同的采样，空分复用可以在一定数量发射天线配置下大幅提高数据速率。MIMO 接收器从接收信号中复原调制符号的最优估计。本书中涉及的估计算法为以下三种：迫零（ZF）、最小均方误差（MMSE）和球形解码器（SD）。下面，我们将详细讨论层映射、预编码，和 MIMO 接收器操作。

层映射对应不同天线将信号数据流分割为子数据流。下面的 MATLAB 函数展示了调制数据流如何从一两个码字映射到 LTE 定义的层（天线端口）上。在这一步，假设进行满秩传输。则层数等于发射天线数。函数的输入为第一个和第二个码字的调制符号流（in1 和 in2），PDSCH 参数结构（prmLTEPDSCH）。根据码字数和层数的不同，函数重组输入符号流生成输出信号（out）。输入信号为 2D 矩阵，它的第二阶等于层数。

Algorithm

MATLAB function

```
function out = LayerMapper(in1, in2, prmLTEPDSCH)
% Layer mapper for spatial multiplexing.
%
%#codegen
% Assumes the incoming codewords are of the same length.
q = prmLTEPDSCH.numCodeWords;          % Number of codewords
v = prmLTEPDSCH.numLayers;             % Number of layers
inLen1 = size(in1, 1);
inLen2 = size(in2, 1);
switch q
   case 1 % Single codeword
      % for numLayers = 1,2,3,4
      out = reshape(in1, v, inLen1/v).';
   case 2 % Two codewords
      switch v
         case 2
            out = complex(zeros(inLen1, v));
            out(:,1) = in1;
            out(:,2) = in2;
```

```
        case 4
            out = complex(zeros(inLen1/2, v));
            out(:,1:2) = reshape(in1, 2, inLen1/2).';
            out(:,3:4) = reshape(in2, 2, inLen2/2).';
        case 6
            out = complex(zeros(inLen1/3, v));
            out(:,1:3) = reshape(in1, 3, inLen1/3).';
            out(:,4:6) = reshape(in2, 3, inLen2/3).';
        case 8
            out = complex(zeros(inLen1/4, v));
            out(:,1:4) = reshape(in1, 4, inLen1/4).';
            out(:,5:8) = reshape(in2, 4, inLen2/4).';
        otherwise
            assert(false, 'This mode is not implemented yet.');
    end
end
```

6.7.5.1 预编码

预编码对每个子数据流数据进行线性变换，以提高总接收端性能。下面的 MATLAB 函数展示层映射之后的多天线子数据流如何进行预编码，之后进行资源元素映射生成资源网格。函数输入为天线端口上的原始调制符号（in），预编码索引（cbIdx），和 PDSCH 参数结构体（prmLTEPDSCH）。首先，我们通过调用 SpatialMuxPrecoder 函数计算正交预编码矩阵（Wn）。然后我们将预编码矩阵和输入向量相乘得到预编码输出（out）。在给定采样时间对所有发射天线采样得到输入向量。

Algorithm

MATLAB function

```
function [out, Wn] = SpatialMuxPrecoder(in, prmLTEPDSCH, cbIdx)
% Precoder for PDSCH spatial multiplexing
%#codegen
% Assumes the incoming codewords are of the same length
v = prmLTEPDSCH.numLayers;          % Number of layers
numTx = prmLTEPDSCH.numTx;          % Number of Tx antennas
% Compute the precoding matrix
Wn = PrecoderMatrix(cbIdx, numTx, v);
% Initialize the output
out = complex(zeros(size(in)));
inLen = size(in, 1);
% Apply the relevant precoding matrix to the symbol over all layers
for n = 1:inLen
    temp = Wn * (in(n, :).');
    out(n, :) = temp.';
end
```

PrecoderMatrix 函数由码书内的值计算预编码矩阵（Wn）。码书值可参阅参考文献［7］。函数输入为预编码索引（cdIdx）、发射天线数（numTx），和层数（v）。忽略预编码是否是开环或闭环，在收发端每个子帧上同时选择一个普通码书索引，对两天线配置来说，有效码书索引为 0 ~ 3，对四天线配置来说有效索引为 0 ~ 15。注意对两天线传输且层数也为 2 的情况，有效码书索引仅有 1 和 2。同样注意对四天线配置来说，预编码矩阵从 1 ×4 码书向量中计算得来，且矩阵运算得到的预编码矩阵对任意给定索引都为正交矩阵。

Algorithm

MATLAB function

```
function Wn = PrecoderMatrix(cbIdx, numTx, v)
% LTE Precoder for PDSCH spatial multiplexing.
%#codegen
% v       = Number of layers
% numTx  = Number of Tx antennas
switch numTx
  case 2
    Wn = complex(ones(numTx, v));
    switch v
      case 1
        a=(1/sqrt(2));
        codebook = [a,a; a,-a; a, 1j*a; a, -1j*a];
        Wn = codebook(cbIdx+1,:);
      case 2
        if cbIdx==1
          Wn = (1/2)*[1 1; 1 -1];
        elseif cdIdx==2
          Wn = (1/2)*[1 1; 1j -1j];
        else
          error('Not used. Please try with a different index.');
        end
    end
  case 4
    un = complex(ones(numTx, 1));
    switch cbIdx
      case 0, un = [1 -1 -1 -1].';
      case 1, un = [1 -1j 1 1j].';
      case 2, un = [1 1 -1 1].';
      case 3, un = [1 1j 1 -1j].';
      case 4, un = [1 (-1-1j)/sqrt(2) -1j (1-1j)/sqrt(2)].';
      case 5, un = [1 (1-1j)/sqrt(2) 1j (-1-1j)/sqrt(2)].';
      case 6, un = [1 (1+1j)/sqrt(2) -1j (-1+1j)/sqrt(2)].';
      case 7, un = [1 (-1+1j)/sqrt(2) 1j (1+1j)/sqrt(2)].';
```

```
        case 8, un = [1 -1 1 1].';
        case 9, un = [1 -1j -1 -1j].';
        case 10, un = [1 1 1 -1].';
        case 11, un = [1 1j -1 1j].';
        case 12, un = [1 -1 -1 1].';
        case 13, un = [1 -1 1 -1].';
        case 14, un = [1 1 -1 -1].';
        case 15, un = [1 1 1 1].';
    end
    Wn = eye(4) - 2*(un*un')./(un'*un);
    switch cbIdx % order columns, for numLayers=4 only
        case {2, 3, 14}
            Wn = Wn(:, [3 2 1 4]);
        case {6, 7, 10, 11, 13}
            Wn = Wn(:, [1 3 2 4]);
    end
    Wn = Wn./sqrt(v);
end
```

6.7.5.2 MIMO 接收器

MIMO 接收器反向预编码和复原调制符号最优估计的处理。如 MIMO 信道建模，在每个时序 n 上接收信号向量 $\vec{Y}$（n）可以表示为发射信号 $\vec{X}$（n）与信道矩阵 H（n）的积，再与高斯噪声 $\vec{n}$（n）相加。在本书中，我们只考虑 2×2 或 4×4 正方矩阵的信道矩阵以简化讨论。该结果可简单扩展到非正方矩阵，其矩阵求反为矩阵伪逆运算。

例如，对于 4×4 MIMO 配置，在任意子帧和任意时间点，接收向量 $\vec{Y}$ 可表示为

$$\vec{Y}=\begin{bmatrix}y_1\\y_2\\y_3\\y_4\end{bmatrix}=\begin{bmatrix}h_{1,1}&\cdots&h_{1,4}\\\vdots&\ddots&\vdots\\h_{4,1}&\cdots&h_{4,4}\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}+\begin{bmatrix}n_1\\n_2\\n_3\\n_4\end{bmatrix}\tag{6.24}$$

MIMO 接收器运算是解出调制发射符号$\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}$的最优估计并用接收符号$\begin{bmatrix}y_1\\y_2\\y_3\\y_4\end{bmatrix}$表

示。因 AWGN 为随机噪声，噪声向量$\begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{bmatrix}$的实效值不可知。我们只能对每个接收天线估计噪声方差。故，AWGN 的影响已完全包含在接受向量之中。

定义接收信号如下，$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} - \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{bmatrix}$，我们改写方程（6.24）得到：

$$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,4} \\ \vdots & \ddots & \vdots \\ h_{4,1} & \cdots & h_{4,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{6.25}$$

在本节中我们介绍三种最流行的 MIMO 均衡设计，它们得到调制发射符号$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$最优估计的方法为

1）ZF 均衡器：方程两侧对信道矩阵 $\boldsymbol{H} = \begin{bmatrix} h_{1,1} & \cdots & h_{1,4} \\ \vdots & \ddots & \vdots \\ h_{4,1} & \cdots & h_{4,4} \end{bmatrix}$求反。我们下面会看到，ZF 均衡器能抑制非相关性噪声，特别是针对低 SNR 传输环境。

2）MMSE 均衡器：最小化发射向量$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$和其估计$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$的均方误差。MMSE 均衡考虑了 AWGN 噪声和逆矩阵噪声方差带来的偏差影响。MMSE 均衡器在控制重构误差方面拥有比 ZF 更好的性能。

3）SD 均衡器：我们的目标是得到方程（6.25）的最大似然解。SD 算法需要已知所有发射天线的调制方案。它结合 MIMO 均衡、软判决译码和最大后验概

率估计，得到最可能生成接收信号$\begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix}$的发射字节的对数似然比（Log - Likelihood Ratio，LLR）。

下面的函数执行 MIMO 接收器操作，它的输入为接收信号（in）、信道矩阵（chEst），PDSCH 参数结构体（prmLTE），预编码器矩阵（Wn）。根据均衡模式的定义（prmLTE. Eqmode），可以选择使用 ZF、MMSE，或 SD。函数最后得到输出信号（y）。

我们下面会讨论每种均衡接收器算法。每一种算法都有独特的办法反向层映射、预编码、和 MIMO 信道操作过程。ZF 和 MMSE 帮助得到发射调制符号的估计值。SD 并不输出调制符号的估计值，而是输出可能生成调制符号的源比特。

Algorithm

MATLAB function

```
function y = MIMOReceiver(in, chEst, prmLTE, nVar, Wn)
%#codegen
switch prmLTE.Eqmode
   case 1 % ZF receiver
      y = MIMOReceiver_ZF(in, chEst, Wn);
   case 2 % MMSE receiver
     y = MIMOReceiver_MMSE(in, chEst, nVar, Wn);
   case 3 % Sphere Decoder
      y = MIMOReceiver_SphereDecoder(in, chEst, prmLTE, nVar, Wn);
   otherwise
      error('Function MIMOReceiver: ZF, MMSE, Sphere decoder are only
supported MIMO detectors');
end
```

ZF 接收器

下面的 MATLAB 函数中 MIMO 接收器用 ZF 接收器消除 MIMO 信道和多天线传输效应的影响。函数输入为接收信号（in），2D 信道矩阵（chEst）和该子帧的预编码器矩阵（Wn）。函数输出（y）为 ZF 均衡得到的调制符号估计。ZF 均衡过程中，我们单纯求反信道矩阵并乘以接收信号。因发射信号向量在发射端也进行了预编码，故在 MIMO 接收器我们需要将均衡向量与逆预编码矩阵相乘。

Algorithm

MATLAB function

```
function y = MIMOReceiver_ZF(in, chEst, Wn)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the ZF detector.
% Get params
numData = size(in, 1);
y = complex(zeros(size(in)));
iWn = inv(Wn);
%% ZF receiver
for n = 1:numData
    h = squeeze(chEst(n, :, :)); % numTx x numRx
    h = h.';                     % numRx x numTx
    Q = inv(h);
    x = Q * in(n, :).';%#ok
    tmp = iWn * x; %#ok
    y(n, :) = tmp.';
end
```

MMSE 接收器

MMSE 接收器减小误差信号功率 $e(n)$，误差信号功率的定义为均衡信号 $X(n)$ 和原始调制符号 $X(n)$ 之差。我们定义 G 为变换接收信号 $Y(n)$ 的最优均衡器。则误差信号可以表示为：

$$e(n) = \hat{X}(n) - X(n) = GY(n) - X(n) \tag{6.26}$$

现在，我们用发射信号和 MIMO 信道矩阵 $\boldsymbol{H}$ 表示接收信号：

$$Y(n) = \boldsymbol{H}X(n) + n(n) \tag{6.27}$$

假设信道矩阵 $\boldsymbol{H}$ 和均衡器矩阵 $\boldsymbol{G}$ 都为正方矩阵，则误差信号可以表示为：

$$e(n) = \boldsymbol{G}Y(n) - X(n) = \boldsymbol{G}(\boldsymbol{H}X(n) + n(n)) - X(n) = (\boldsymbol{GH} - \boldsymbol{I})X(n) + \boldsymbol{G}n(n) \tag{6.28}$$

对上式，我们可以套用 Wiener 滤波模型求误差信号最小期望值，我们可以得到 MMSE 最优均衡器为：

$$\boldsymbol{G}_{mmse} = \boldsymbol{H}^H(\boldsymbol{HH}^H + \sigma_n^2\boldsymbol{I}_n)^{-1} \tag{6.29}$$

式中　$\boldsymbol{H}^H$ 表示 Hermitian 信道矩阵 $\boldsymbol{H}$；σ_n^2 表示信道噪声方差；$\boldsymbol{I}_n$ 表示阶数为发射天线数的特征矩阵。

下面的 MATLAB 函数为一个使用 MMSE 均衡器的 MIMO 接收器。函数输入为接收信号（in），2D 信道矩阵（chEst），和预编码器矩阵（Wn）。函数用 MMSE 均衡方法输出调制符号估计（y）。我们对每个接收向量在采样时间 n，用

MMSE 均衡器公式计算均衡器矩阵（Q），并乘以接收向量。为了反向预编码操作，我们同样需要将均衡之后的向量乘以逆预编码矩阵。

Algorithm

MATLAB function

```
function y = MIMOReceiver_MMSE(in, chEst, nVar, Wn)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the MMSE detector.
% Get params
numLayers = size(Wn,1);
% noisFac = numLayers*diag(nVar);
noisFac = diag(nVar);
numData = size(in, 1);
y = complex(zeros(size(in)));
iWn = inv(Wn);
%% MMSE receiver
for n = 1:numData
   h = chEst(n, :, :);                        % numTx x numRx
   h = reshape(h(:), numLayers, numLayers).'; % numRx x numTx
   Q = (h'*h + noisFac)\h';
   x = Q * in(n, :).';
   tmp = iWn * x;
   y(n, :) = tmp.';
end
```

SD 接收器

SD 接收器对 MIMO 方程求 ML 解。对一个给定采样时间的给定 MIMO 信道方程，有

$$Y = \boldsymbol{H}X + n \tag{6.30}$$

SD 对发射调制符号 $\hat{X}_{ML}$求 ML 估计，即

$$\hat{X}_{ML} = \arg\min \| Y - \boldsymbol{H}X \|^2 \tag{6.31}$$

式中　$X \in \Omega$，Ω 为给定 X 元素的复平面星座点集合。SD 算法有效利用了调制方案以及实际的星座图和调制器符号映射。它结合了 MIMO 均衡和软判决译码，并最大化后验概率测量以产生输出。SD 的输出为最有可能生成调制符号的源比特的 LLR。通信系统工具箱的 comm. SphereDecoder 系统对象可实现 SD 算法。该系统对象构成的 ML 接收器凭借软球形译码器（SSD）降低了计算复杂度。

下面的 MATLAB 函数实现了使用 SD 的 MIMO 接收器。函数输入为接收信号（in），3D 信道矩阵（chEst），和 PDSCH 参数结构体（prmLTE），噪声方差向量（nVar），以及预编码器矩阵（Wn）。函数用球形译码器均衡方法输出调制符号

估计（y）。首先，我们将通过逆预编码矩阵对信道矩阵进行变换。然后我们用 comm. SphereDecoder 系统对象执行最大似然（ML）估计球形译码操作。

Algorithm

MATLAB function

```
function [y, bittable] = MIMOReceiver_SphereDecoder(in, chEst, prmLTE, nVar, Wn)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the Sphere detector.
% Soft-Sphere Decoder
symMap=prmLTE.SymbolMap;
numBits=prmLTE.Qm;
constell=prmLTE.Constellation;
bittable = de2bi(symMap, numBits, 'left-msb');
iWn=Wn.';
nVar1=(-1/mean(nVar));
persistent SphereDec
if isempty(SphereDec)
   % Soft-Sphere Decoder
   SphereDec = comm.SphereDecoder('Constellation', constell,...
      'BitTable', bittable, 'DecisionType', 'Soft');
end
% SSD receiver
temp = complex(zeros(size(chEst)));
% Account for precoding
for n = 1:size(chEst,1)
   temp(n, :, :) = iWn * squeeze(chEst(n, :, :));
end
hD = temp;
y = nVar1 * step(SphereDec, in, hD);
```

6.7.6 下行链路传输模式 4

在本节中，我们关注 LTE 标准中 MIMO 最具突破性、可实现最高数据速率的模式：模式 4。该模式采用空分复用预编码和闭环信道反馈。在低移动率情况下，闭环反馈信道质量可以提高性能。我们会在第 7 章实现真实的闭环反馈操作。在本章中，我们在本章先用静态预编码矩阵进行讲解，作为下一章更好了解闭环适应性预编码技术的垫脚石。

我们会分两部分讲解模式 4：

1）单码字：在 DLSCH 只生成一个码字，并在 PDSCH 处理；

2）双码字：在 DLSCH 生成两个不同的码字，并通过层映射复用，进行预编码、资源元素映射，及 OFDM 传输。

6.7.6.1 单码字

下面的 MATLAB 函数实现 LTE 传输模式 4 的发射端、接收端和信道模型，使用单码字空分复用。我们配置 2×2 和 4×4 两种多天线收发阵列情况。本例中关键组件如下：

1）生成单子帧载荷数据（一个传输块）；

2）DLSCH 处理，如前文所述；

3）PDSCH 发射端处理，包括比特级绕码、数据调制、层映射、二或四天线预编码以及空分复用预编码、资源元素映射，和 OFDM 信号生成；

4）信道建模，包括 MIMO 衰落信道附加 AWGN 信道；

5）PDSCH 接收端处理，包括 OFDM 信号接收生成资源网格、资源元素反映射分隔 CSR 信号、信道估计、MIMO 接收与层反映射、软判决译码、解绕码、和 DLSCH 译码。

Algorithm

MATLAB script

```
function [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr_ref]...
   = commlteMIMO_SM_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl)
%% TX
persistent hPBer1
if isempty(hPBer1), hPBer1=comm.ErrorRate; end;
%  Generate payload
dataIn = genPayload(nS,  prmLTEDLSCH.TBLenVec);
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
[data, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH, prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
% Map modulated symbols  to layers
numTx=prmLTEPDSCH.numTx;
LayerMapOut = LayerMapper(modOut, [], prmLTEPDSCH);
usedCbIdx = prmMdl.cbIdx;
% Precoding
[PrecodeOut, Wn] = lteSpatialMuxPrecoder(LayerMapOut, prmLTEPDSCH, usedCbIdx);
% Generate Cell-Specific Reference (CSR) signals
csr = CSRgenerator(nS, numTx);
csr_ref=complex(zeros(2*prmLTEPDSCH.Nrb, 4, numTx));
for m=1:numTx
   csr_pre=csr(1:2*prmLTEPDSCH.Nrb,:,:,m);
   csr_ref(:,:,m)=reshape(csr_pre,2*prmLTEPDSCH.Nrb,4);
end
```

```
% Resource grid filling
txGrid = REmapper_mTx(PrecodeOut, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
%% Channel
% MIMO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
% Add AWG noise
sigPow = 10*log10(var(rxFade));
nVar = 10.^(0.1.*(sigPow-snrdB));
rxSig =  AWGNChannel(rxFade, nVar);
%% RX
% OFDM Rx
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = REdemapper_mTx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
   chEst = ChanEstimate_mTx(prmLTEPDSCH, csrRx,  csr_ref, prmMdl.chEstOn);
   hD     = ExtChResponse(chEst, idx_data, prmLTEPDSCH);
else
   idealChEst = IdChEst(prmLTEPDSCH, prmMdl, chPathG);
   hD =  ExtChResponse(idealChEst, idx_data, prmLTEPDSCH);
end
% Frequency-domain equalizer
if (numTx==1)
   % Based on Maximum-Combining Ratio (MCR)
   yRec = Equalizer_simo(dataRx, hD, nVar, prmLTEPDSCH.Eqmode);
else
   % Based on Spatial Multiplexing
   yRec = MIMOReceiver(dataRx, hD, prmLTEPDSCH, nVar, Wn);
end
% Demap received codeword(s)
[cwOut, ~] = LayerDemapper(yRec, prmLTEPDSCH);
if prmLTEPDSCH.Eqmode < 3
   % Demodulate
   demodOut = DemodulatorSoft(cwOut, prmLTEPDSCH.modType, mean(nVar));
else
   demodOut = cwOut;
end
   % Descramble received codeword
   rxCW =  lteDescramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes - CB segmentation, turbo decoding, rate dematching
[decTbData1, ~,~] = lteTbChannelDecoding(nS, rxCW, Kplus1, C1,  prmLTEDLSCH,
prmLTEPDSCH);
% Transport block CRC detection
[dataOut, ~] = CRCdetector(decTbData1);
end
```

收发端模型的结构

下面的 MATLAB 测试脚本调用 MIMO 收发端函数 commlteMIMO。首先，调用初始化函数（commlteMIMO _ initialize）设置所有相关参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。然后在一个 While 循环内调用 MIMO 收发端函数（commlteMIMO _ SM _ step）进行子帧处理。最后，计算 BER 并调用可视化程序显示信道响应和均衡前后的调制星座图。

Algorithm

MATLAB script

```
% Script for MIMO LTE (mode 4)
%
% Single codeword transmission
%
clear all
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl corrLvl chEstOn numCodeWords enPMIfback cbIdx snrdB
maxNumErrs maxNumBits
%%
disp('Simulating the LTE Mode 3: Multiple Tx & Rx antrennas with Spatial Multiplexing');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
hPBer = comm.ErrorRate;
snrdB=prmMdl.snrdB;
maxNumErrs=prmMdl.maxNumErrs;
maxNumBits=prmMdl.maxNumBits;
%% Simulation loop
tic;
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
  [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
    commlteMIMO_SM_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
  % Calculate  bit errors
  Measures = step(hPBer, dataIn, dataOut);
   % Visualize constellations and spectrum
  if (visualsOn && prmLTEPDSCH.Eqmode~=3)
    zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
  end;
  % Update subframe number
  nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
```

```
end
disp(Measures);
toc;
```

验证收发端性能

通过执行 MIMO 收发器模型（commlteMIMO）的 MATLAB 测试脚本，我们可以通过观察各个信号评估系统性能。仿真涉及的参数总结在下面的 MATLAB 脚本（commlteMIMO _ params）。这些参数设定与 6.7.1 节相同。参数可以反映单码字空分复用 MIMO 模式 4 的不同之处，不使用预编码矩阵反馈，以及用在 MIMO 接收器进行 MMSE 均衡处理。本仿真处理一百万用户数据比特，AWGN 信道的 SNR 为 16dB，可视化输出。

Algorithm

MATLAB script

```
% PDSCH
txMode         = 4;  % Transmission mode one of {1, 2, 4}
numTx          = 2;   % Number of transmit antennas
numRx          = 2;   % Number of receive antennas
chanBW         = 4;        % [1,2,3,4,5,6] maps to [1.4, 3, 5, 10, 15, 20]MHz
contReg        = 1;        % {1,2,3} for >=10MHz, {2,3,4} for <10Mhz
modType        = 2;        % [1,2,3] maps to ['QPSK','16QAM','64QAM']
% DLSCH
cRate          = 1/3; % Rate matching target coding rate
maxIter        = 6;    % Maximum number of turbo decoding terations
fullDecode     = 0;   % Whether "full" or "early stopping" turbo decoding is performed
% Channel model
chanMdl         = 'frequency-selective-high-mobility';
% one of {'flat-low-mobility', 'flat-high-mobility','frequency-selective-low-mobility',
% 'frequency-selective-high-mobility', 'EPA 0Hz', 'EPA 5Hz', 'EVA 5Hz', 'EVA 70Hz'}
corrLvl         = 'Medium';
% Simulation parameters
Eqmode         = 2;    % Type of equalizer used [1,2,3] for ['ZF', 'MMSE','Sphere Decoder']
chEstOn         = 1;        % use channel estimation or ideal channel
snrdB           = 16;   % Signal to Noise Ratio in dB
maxNumErrs      = 1e6; % Maximum number of errors found before simulation stops
maxNumBits      = 1e6;  % Maximum number of bits processed before simulation stops
visualsOn       = 1;     % Whether to visualize channel response and constellations
numCodeWords  = 1; % Number of codewords in PDSCH
enPMIfback      = 0;    % Enable/Disable Precoder Matrix Indicator (PMI) feedback
cbIdx           = 1;     % Initialize PMI index
```

图 6. 12 表示了双天线中每一个天线上一个子帧的用户数据均衡前（第一行）后（第二行）星座图。可以看到均衡器补偿了信道衰落的影响，使星座图更接近于 16QAM。

图 16. 3 表示了双天线中每一个天线上一个子帧的用户数据频谱。图中可以

图 6.12　LTE 模式：用户数据在均衡前后的 MIMO 空分复用星座图

看到发射信号以及接收信号均衡前后的频谱。均衡前（可看到频率选择性衰落）的接收信号在单码字空分复用（可看到更平坦频谱）后，更接近发射信号频谱。

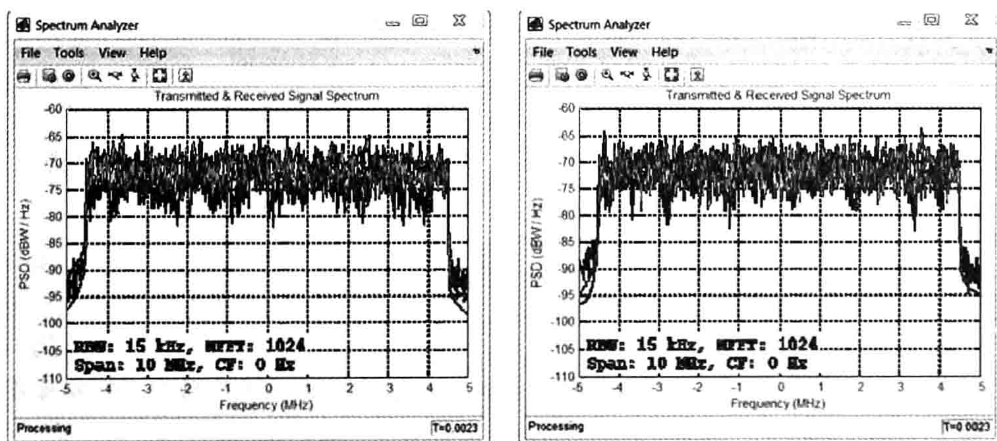


图 6.13　单码字 MIMO 发射信号和均衡前后接收信号的空分复用频谱

BER 测量

为了验证收发端的 BER 性能，我们创建了 commlteMIMO _ test _ timing _ ber 的测试脚本，初始化 LTE 系统变量，并在循环中遍历 SNR 值调用 commlteMIMO _ fcn 计算相应的 BER。

Algorithm

MATLAB script: commlteMIMO_test_timing_ber

```
% Script for MIMO LTE (mode 4)
%
% Single codeword transmission only
%
clear all
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_params;
maxNumErrs=5e7;
maxNumBits=5e7;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl corrLvl chEstOn numCodeWords enPMIfback cbIdx snrdB
maxNumErrs maxNumBits
%%
disp('Simulating the LTE Mode 3: Multiple Tx & Rx antrennas with Spatial Multiplexing');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
%% Geerate code and setup parallelism
disp('Generating code for commlteMIMO_fcn.m ...');
arg1=coder.Constant(prmLTEPDSCH);
arg2=coder.Constant( prmLTEDLSCH);
arg3=coder.Constant(prmMdl);
codegen commlteMIMO_fcn -args {16, arg1, arg2, arg3} -report
disp('Done.');
parallel_setup;
%%
MaxIter=8;
snr_vector=getSnrVector(prmLTEPDSCH.modType, MaxIter);
ber_vector=zeros(size(snr_vector));
maxNumBits=prmMdl.maxNumBits;
tic;
parfor n=1:MaxIter
   fprintf(1,'Iteration %2d out of %2d:  Processing %10d bits. SNR = %3d\n', ...
      n, MaxIter, maxNumBits, snr_vector(n));
   [ber, ~] = commlteMIMO_fcn_mex(snr_vector(n), prmLTEPDSCH, prmLTEDLSCH,
```

```
prmMdl);
    ber_vector(n)=ber;
end;
toc;
semilogy(snr_vector, ber_vector);
title('BER - commlteMIMO SM');xlabel('SNR (dB)');ylabel('ber');grid;
```

图 6. 14 所示为收发端 BER 随 SNR 值的变化关系。程序进行 8 次迭代，每次处理 5000 万比特用户数据。

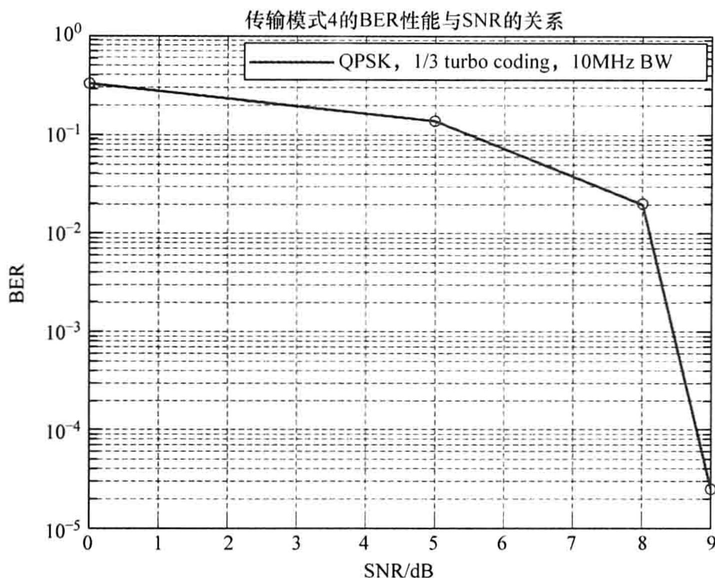


图 6. 14 BER 结果：LTE 模式 4 空分复用单码字（2 × 2）MIMO 信道

6. 7. 6. 2 双码字

下面的 MATLAB 函数实现 LTE 传输模式 4 的发射端、接收端，和信道模型，使用双码字空分复用。收发器结构与单码字类似，除了我们创建和处理一对数据比特并重复处理如 CRC、DLSCH 处理、绕码和调制。层映射将数据变换到层。这一步之后到层反映射之间的步骤与单天线情况类似。层反映射之后的解调、解绕码、CRC 校验，和传输块信道译码（反向 DLSCH 处理）同样需要处理一对比特。

Algorithm

MATLAB function

```
function [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr_ref]...
    = commlteMIMO_SM2_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl)
%% TX
```

```
persistent hPBer1
if isempty(hPBer1), hPBer1=comm.ErrorRate; end;
% Generate payload
dataIn1 = genPayload(nS, prmLTEDLSCH.TBLenVec);
dataIn2 = genPayload(nS, prmLTEDLSCH.TBLenVec);
dataIn=[dataIn1;dataIn2];
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn1);
tbCrcOut2 =CRCgenerator(dataIn2);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
[data1, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH,
prmLTEPDSCH);
[data2, Kplus2, C2] = lteTbChannelCoding(tbCrcOut2, nS, prmLTEDLSCH,
prmLTEPDSCH);
%Scramble codeword
scramOut1 = lteScramble(data1, nS, 0, prmLTEPDSCH.maxG);
scramOut2 = lteScramble(data2, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut1 = Modulator(scramOut1, prmLTEPDSCH.modType);
modOut2 = Modulator(scramOut2, prmLTEPDSCH.modType);
% Map modulated symbols to layers
numTx=prmLTEPDSCH.numTx;
LayerMapOut = LayerMapper(modOut1, modOut2, prmLTEPDSCH);
usedCbIdx = prmMdl.cbIdx;
% Precoding
[PrecodeOut, Wn] = lteSpatialMuxPrecoder(LayerMapOut, prmLTEPDSCH, usedCbIdx);
% Generate Cell-Specific Reference (CSR) signals
csr = CSRgenerator(nS, numTx);
csr_ref=complex(zeros(2*prmLTEPDSCH.Nrb, 4, numTx));
for m=1:numTx
   csr_pre=csr(1:2*prmLTEPDSCH.Nrb,:,:,m);
   csr_ref(:,:,m)=reshape(csr_pre,2*prmLTEPDSCH.Nrb,4);
end
% Resource grid filling
txGrid = REmapper_mTx(PrecodeOut, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
%% Channel
% MIMO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
% Add AWG noise
sigPow = 10*log10(var(rxFade));
nVar = 10.^(0.1.*(sigPow-snrdB));
rxSig = AWGNChannel(rxFade, nVar);
%% RX
% OFDM Rx
```

```
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = REdemapper_mTx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
    chEst = ChanEstimate_mTx(prmLTEPDSCH, csrRx,  csr_ref, prmMdl.chEstOn);
    hD     = ExtChResponse(chEst, idx_data, prmLTEPDSCH);
else
    idealChEst = IdChEst(prmLTEPDSCH, prmMdl, chPathG);
    hD =  ExtChResponse(idealChEst, idx_data, prmLTEPDSCH);
end
% Frequency-domain equalizer
if (numTx==1)
    % Based on Maximum-Combining Ratio (MCR)
    yRec = Equalizer_simo(dataRx, hD, nVar, prmLTEPDSCH.Eqmode);
else
    % Based on Spatial Multiplexing
    yRec = MIMOReceiver(dataRx, hD, prmLTEPDSCH, nVar, Wn);
end
% Demap received codeword(s)
[cwOut1, cwOut2] = LayerDemapper(yRec, prmLTEPDSCH);
if prmLTEPDSCH.Eqmode < 3
    % Demodulate
    demodOut1 = DemodulatorSoft(cwOut1, prmLTEPDSCH.modType, mean(nVar));
    demodOut2 = DemodulatorSoft(cwOut2, prmLTEPDSCH.modType, mean(nVar));
else
    demodOut1 = cwOut1;
    demodOut2 = cwOut2;
end
% Descramble received codeword
rxCW1 =  lteDescramble(demodOut1, nS, 0, prmLTEPDSCH.maxG);
rxCW2 =  lteDescramble(demodOut2, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes  CB segmentation, turbo decoding, rate dematching
[decTbData1, ~,~] = lteTbChannelDecoding(nS, rxCW1, Kplus1, C1,  prmLTEDLSCH,
prmLTEPDSCH);
[decTbData2, ~,~] = lteTbChannelDecoding(nS, rxCW2, Kplus2, C2,  prmLTEDLSCH,
prmLTEPDSCH);
% Transport block CRC detection
[dataOut1, ~] = CRCdetector(decTbData1);
[dataOut2, ~] = CRCdetector(decTbData2);
dataOut=[dataOut1;dataOut2];
end
```

收发端模型结构

下面的 MATLAB 测试脚本调用 MIMO 收发端函数 commlteMIMO。首先，调用初始化函数（commlteMIMO _ initialize）设置所有相关参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。然后在一个 While 循环内调用 MIMO 收发

端函数（commlteMIMO _ SM2 _ step）进行子帧处理。最后，计算 BER 并调用可视化程序显示信道响应和均衡前后的调制星座图。

Algorithm

MATLAB script

```
% Script for MIMO LTE (mode 4)
%
% Two codeword transmission
%
clear all
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_ params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl corrLvl chEstOn numCodeWords enPMIfback cbIdx snrdB
maxNumErrs maxNumBits
%%
disp('Simulating the LTE Mode 3: Multiple Tx & Rx antennas with Spatial Multiplexing');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
hPBer = comm.ErrorRate;
snrdB=prmMdl.snrdB;
maxNumErrs=prmMdl.maxNumErrs;
maxNumBits=prmMdl.maxNumBits;
%% Simulation loop
tic;
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
  [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
     commlteMIMO_SM2_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
   % Calculate  bit errors
   Measures = step(hPBer, dataIn, dataOut);
    % Visualize constellations and spectrum
   if (visualsOn && prmLTEPDSCH.Eqmode~=3)
      zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
   end;
   % Update subframe number
   nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
end
disp(Measures);
toc;
```

验证收发端性能

通过执行 MIMO 收发器模型（commlteMIMO）的 MATLAB 测试脚本，我们可以通过观察各个信号评估系统性能。仿真涉及的参数总结在下面的 MATLAB 脚本（commlteMIMO _ params）。这些参数设定与 6.7.1 节相同。参数可以反映双码字空分复用 MIMO 模式 4 的不同之处，不使用预编码矩阵反馈，以及用在 MIMO 接收器进行 MMSE 均衡处理。本仿真处理一百万用户数据比特，AWGN 信道的 SNR 为 16dB，可视化输出。

Algorithm

MATLAB script

```
% PDSCH
txMode          = 4;  % Transmission mode one of {1, 2, 4}
numTx           = 2;  % Number of transmit antennas
numRx           = 2;  % Number of receive antennas
chanBW          = 4;      % [1,2,3,4,5,6] maps to [1.4, 3, 5, 10, 15, 20]MHz
contReg         = 1;      % {1,2,3} for >=10MHz, {2,3,4} for <10Mhz
modType         = 2;      % [1,2,3] maps to ['QPSK','16QAM','64QAM']
% DLSCH
cRate           = 1/3; % Rate matching target coding rate
maxIter         = 6;   % Maximum number of turbo decoding terations
fullDecode      = 0;   % Whether "full" or "early stopping" turbo decoding is performed
% Channel model
chanMdl         = 'frequency-selective-high-mobility';
% one of {'flat-low-mobility', 'flat-high-mobility','frequency-selective-low-mobility',
% 'frequency-selective-high-mobility', 'EPA 0Hz', 'EPA 5Hz', 'EVA 5Hz', 'EVA 70Hz'}
corrLvl         = 'Medium';
% Simulation parameters
Eqmode          = 2;   % Type of equalizer used [1,2,3] for ['ZF', 'MMSE','Sphere Decoder']
chEstOn         = 1;       % use channel estimation or ideal channel
snrdB           = 16;  % Signal to Noise Ratio in dB
maxNumErrs      = 1e6; % Maximum number of errors found before simulation stops
maxNumBits      = 1e6; % Maximum number of bits processed before simulation stops
visualsOn       = 1;   % Whether to visualize channel response and constellations
numCodeWords    = 2; % Number of codewords in PDSCH
enPMIfback      = 0;   % Enable/Disable Precoder Matrix Indicator (PMI) feedback
cbIdx           = 1;   % Initialize PMI index
```

图 6.15 表示了双天线中每一个天线上一个子帧的用户数据均衡前（第一行）后（第二行）星座图。可以看到均衡器补偿了信道衰落的影响，使星座图更接近于 16QAM。

图 6.16 表示了双天线中每一个天线上一个子帧的用户数据频谱。图中可以看到发射信号以及接收信号均衡前后的频谱。均衡前的接收信号在单码字空分复

用后，更接近发射信号频谱。

图 6.15 LTE 模型：MIMO 空分复用双码字均衡前后星座图

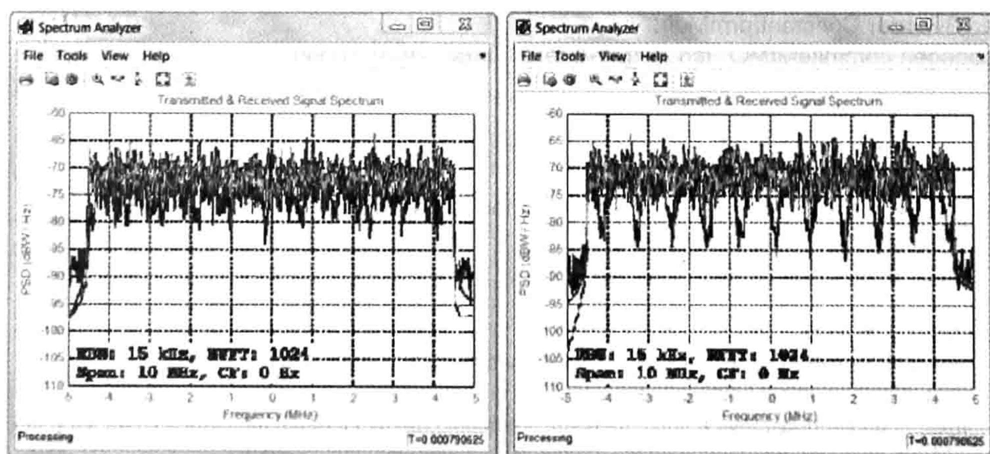


图 6.16 双码字 MIMO 发射信号和均衡前后接收信号的空分复用频谱

BER 测量

为了验证收发端的 BER 性能，我们创建了 commlteMIMO _ test _ timing _ ber 的测试脚本，初始化 LTE 系统变量，并在循环中遍历 SNR 值调用 commlteMIMO _ fcn 计算相应的 BER。其结果与图 6. 14 中的单码字结果非常接近。

Algorithm

MATLAB script: commlteMIMO_test_timing_ber

```
% Script for MIMO LTE (mode 4)
%
% Single codeword transmission only
%
clear all
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_params;
maxNumErrs=5e7;
maxNumBits=5e7;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl corrLvl chEstOn numCodeWords enPMIfback cbIdx snrdB
maxNumErrs maxNumBits
%%
disp('Simulating the LTE Mode 3: Multiple Tx & Rx antrennas with Spatial Multiplexing');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
%% Geerate code and setup parallelism
disp('Generating code for commlteMIMO_fcn.m ...');
arg1=coder.Constant(prmLTEPDSCH);
arg2=coder.Constant( prmLTEDLSCH);
arg3=coder.Constant(prmMdl);
codegen commlteMIMO_fcn -args {16, arg1, arg2, arg3} -report
disp('Done.');
parallel_setup;
%%
MaxIter=8;
snr_vector=getSnrVector(prmLTEPDSCH.modType, MaxIter);
ber_vector=zeros(size(snr_vector));
maxNumBits=prmMdl.maxNumBits;
tic;
parfor n=1:MaxIter
   fprintf(1,'Iteration %2d out of %2d:  Processing %10d bits. SNR = %3d\n', ...
      n, MaxIter, maxNumBits, snr_vector(n));
   [ber, ~] = commlteMIMO_fcn_mex(snr_vector(n), prmLTEPDSCH, prmLTEDLSCH,
prmMdl);
```

```
    ber_vector(n)=ber;
end;
toc;
semilogy(snr_vector, ber_vector);
title('BER - commlteMIMO SM');xlabel('SNR (dB)');ylabel('ber');grid;
```

6.7.7 开环空分复用

下面的 MATLAB 函数实现大范围循环延迟分集空分复用算法即 LTE 标准中的 MIMO 传输模式 3。开环预编码应用与高移动率情况且不需要依赖用户终端的循环矩阵指示（PMI）。当移动终端快速移动时，由于不能依靠先前帧得到的信道暂态反馈求当前信道质量，开环空分复用不需要从基站向移动终端传输预编码矩阵的额外信息。预编码矩阵已在收发两端提前确定并同时计算。

6.7.7.1 开环预编码

在开环预编码中，收发端不需要用反馈环相互通信选择码书索引，而是使用预先设定的编码矩阵索引。发射端和接收端根据发射采样同步更新该预设定索引集。

开环预编码传输矩阵的秩可变，可由满秩到最小两层。当秩估计的结果显示为单层时，我们从空分复用切模式切换到发射分集。对传输模式 3，当秩为 1 时，两天线使用 SFBC 而四天线使用 SFBC/FSTD。

PrecoderMatrix 函数计算每个输入信号的预编码矩阵（W），对角矩阵（D），和本征矩阵（U）。函数输入为采样索引（n）和层数（v）。码书值根据参考文献［7］设定。

Algorithm

MATLAB function

```
function [W, D, U] = PrecoderMatrix(n, v)
% LTE Precoder for PDSCH spatial multiplexing.
%#codegen
idx=mod(n-1,4);
switch v
   case 1
      W=complex(1,0);
      U=W;D=W;
   case 2
      W=[1 0; 0 1];
      U=(1/sqrt(2))*[1 1;1 exp(-1j*pi)];
      D=[1 0;0 exp(-1j*pi*idx)];
   case 4
      k=1+mod(floor(n/4),4);
      switch k
```

```
        case 1, un = [1 -1 -1 1].';
        case 2, un = [1 -1 1 -1].';
        case 3, un = [1 1 -1 -1].';
        case 4, un = [1 1 1 1].';
    end
    W = eye(4) - 2*(un*un')./(un'*un);
    switch k % order columns
        case 3
            W = W(:, [3 2 1 4]);
        case 2
            W = W(:, [1 3 2 4]);
    end
    a=[0*(0:1:3);2*(0:1:3);4*(0:1:3);6*(0:1:3)];
    U=(1/2)*exp(-1j*pi*a/4);
    b=0:1:3;
    D=diag(exp(-1j*2*pi*idx*b/4));
end
```

下面的 MATLAB 函数进行开环空分复用预编码操作。函数输入为给定层调制符号（in）、预编码索引（cbIdx），和 PDSCH 参数结构体（prmLTEPDSCH）。计算输出（out）分为三个步骤：

1）每个输入采样的处理循环中，PrecoderMatrix 计算输出三个矩阵（W，D，U）；

2）这些矩阵相乘得到传输矩阵（T）；

3）通过输入采样和传输矩阵相乘得到输入向量预编码。

Algorithm

MATLAB function

```
function out = SpatialMuxPrecoder(in, prmLTEPDSCH)
% Precoder for PDSCH spatial multiplexing
%#codegen
% Assumes the incoming codewords are of the same length
v = prmLTEPDSCH.numLayers;              % Number of layers
% Initialize the output
out = complex(zeros(size(in)));
inLen = size(in, 1);
% Apply the relevant precoding matrix to the symbol over all layers
for n = 1:inLen
    % Compute the precoding matrix
    [W, D, U] = PrecoderMatrix(n, v);
    T=W *D*U;
    temp = T* (in(n, :).');
    out(n, :) = temp.';
end
```

6.7.7.2 MIMO 接收器操作

开环空分复用 MIMO 接收器函数与闭环空分复用相似。其输入为接收信号（in）、信道矩阵（chEst），和 PDSCH 参数结构体（prmLTE），以及噪声方差向量（nVar）。根据均衡模式不同（prmLTE. Eqmode），函数可以选择 ZF、MMSR，或 SD 接收器均衡并声称输出信号（y）。

Algorithm

MATLAB function

```
function y = MIMOReceiver_OpenLoop(in, chEst, prmLTE, nVar)
%#codegen
v=prmLTE.numTx;
switch prmLTE.Eqmode
   case 1 % ZF receiver
      y = MIMOReceiver_ZF_OpenLoop(in, chEst, v);
   case 2 % MMSE receiver
     y = MIMOReceiver_MMSE_OpenLoop(in, chEst, nVar, v);
   case 3 % Sphere Decoder
      y = MIMOReceiver_SD_OpenLoop(in, chEst, prmLTE, nVar, v);
   otherwise
      error('Function MIMOReceiver: ZF, MMSE, Sphere decoder are only
supported MIMO detectors');
end
```

ZF 接收器

下面的 MATLAB 函数为使用 ZF 方法的 MIMO 接收器。函数输入为接收信号（in）、2D 信道矩阵（chEst），和子帧层数（v）。依照 ZF 均衡方法，输出估计调制信号（y）。

Algorithm

MATLAB function

```
function y = MIMOReceiver_ZF_OpenLoop(in, chEst, v)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the ZF detector.
% Get params
numData = size(in, 1);
y = complex(zeros(size(in)));
%% ZF receiver
for n = 1:numData
   [W, D, U] = PrecoderMatrixOpenLoop(n, v);
   iWn = (W *D*U)';
   h = squeeze(chEst(n, :, :)); % numTx x numRx
```

```
    h = h.';                % numRx x numTx
    x = h \ (in(n, :).');
    tmp = iWn * x;
    y(n, :) = tmp.';
end
```

MMSE 接收器

下面的 MATLAB 函数为使用 MMSE 方法的 MIMO 接收器。函数的输入与输出与 ZF 算法类似，唯一不同的是其输入另有一项为当前帧的噪声方差（nVar）。

Algorithm

MATLAB function

```
function y = MIMOReceiver_MMSE_OpenLoop(in, chEst, nVar, v)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the MMSE detector.
% noisFac = numLayers*diag(nVar);
noisFac = diag(nVar);
numData = size(in, 1);
y = complex(zeros(size(in)));
%% MMSE receiver
for n = 1:numData
    [W, D, U] = PrecoderMatrixOpenLoop(n, v);
    iWn = (W *D*U)';            % Orthonormal matrix
    h = chEst(n, :, :);          % numTx x numRx
    h = reshape(h(:), v, v).';   % numRx x numTx
    Q = (h'*h + noisFac)\h';
    x = Q * in(n, :).';
    tmp = iWn * x;
    y(n, :) = tmp.';
end
```

SD 接收器

下面的 MATLAB 函数为使用球形译码器方法的 MIMO 接收器。函数的输入与输出及其变量名与 MMSE 接收器相同。

Algorithm

MATLAB function

```
function [y, bittable] = MIMOReceiver_SD_OpenLoop(in, chEst, prmLTE, nVar, v)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the Sphere detector.
```

```
% Soft-Sphere Decoder
symMap=prmLTE.SymbolMap;
numBits=prmLTE.Qm;
constell=prmLTE.Constellation;
bittable = de2bi(symMap, numBits, 'left-msb');
nVar1=(-1/mean(nVar));
persistent SphereDec
if isempty(SphereDec)
   % Soft-Sphere Decoder
   SphereDec = comm.SphereDecoder('Constellation', constell,...
      'BitTable', bittable, 'DecisionType', 'Soft');
end
% SSD receiver
temp = complex(zeros(size(chEst)));
% Account for precoding
for n = 1:size(chEst,1)
   [W, D, U] =PrecoderMatrixOpenLoop(n, v);
   iWn = (W *D*U).';
   temp(n, :, :) =  iWn * squeeze(chEst(n, :, :)) ;
end
hD = temp;
y = nVar1 * step(SphereDec, in, hD);
```

6.7.8 下行链路传输模式 3

第三种下行链路传输模式使用开环空分复用以实现高移动率情况下的传输。下面的 MATLAB 函数包含发射端、接收端以及信道模型，展示单码书空分复用传输模式。发射端和接收端使用多路天线，我们配置设定 2×2 和 4×4 MIMO 天线。本例中的关键组件如下：

1）生成单子帧载荷数据（一个传输块）；

2）DLSCH 处理，如前文所述；

3）PDSCH 发射端处理，包括比特级绕码、数据调制、层映射和二或四天线预编码，以及空分复用预编码、资源元素映射，和 OFDM 信号生成；

4）信道建模，包括 MIMO 衰落信道附加 AWGN 信道；

5）PDSCH 接收端处理，包括 OFDM 信号接收生成资源网格、资源元素反映射分隔 CSR 信号、信道估计、MIMO 接收与层反映射、软判决译码、解绕码和 DLSCH 译码。

Algorithm

MATLAB script

```
function [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr_ref]...
  = commlteMIMO_SM_Mode3_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl)
```

```
%% TX
persistent hPBer1
if isempty(hPBer1), hPBer1=comm.ErrorRate; end;
% Generate payload
dataIn = genPayload(nS, prmLTEDLSCH.TBLenVec);
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
[data, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH, prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
% Map modulated symbols to layers
numTx=prmLTEPDSCH.numTx;
LayerMapOut = LayerMapper(modOut, [], prmLTEPDSCH);
% Precoding
PrecodeOut = SpatialMuxPrecoderOpenLoop(LayerMapOut, prmLTEPDSCH);
% Generate Cell-Specific Reference (CSR) signals
csr = CSRgenerator(nS, numTx);
csr_ref=complex(zeros(2*prmLTEPDSCH.Nrb, 4, numTx));
for m=1:numTx
   csr_pre=csr(1:2*prmLTEPDSCH.Nrb,:,:,m);
   csr_ref(:,:,m)=reshape(csr_pre,2*prmLTEPDSCH.Nrb,4);
end
% Resource grid filling
txGrid = REmapper_mTx(PrecodeOut, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
%% Channel
% MIMO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
% Add AWG noise
sigPow = 10*log10(var(rxFade));
nVar = 10.^(0.1.*(sigPow-snrdB));
rxSig = AWGNChannel(rxFade, nVar);
%% RX
% OFDM Rx
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = REdemapper_mTx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
   chEst = ChanEstimate_mTx(prmLTEPDSCH, csrRx, csr_ref, prmMdl.chEstOn);
   hD    = ExtChResponse(chEst, idx_data, prmLTEPDSCH);
else
   idealChEst = IdChEst(prmLTEPDSCH, prmMdl, chPathG);
   hD = ExtChResponse(idealChEst, idx_data, prmLTEPDSCH);
end
```

```
% Frequency-domain equalizer
if (numTx==1)
   % Based on Maximum-Combining Ratio (MCR)
   yRec = Equalizer_simo(dataRx, hD,mean(nVar), prmLTEPDSCH.Eqmode);
else
   % Based on Spatial Multiplexing
   yRec = MIMOReceiver_OpenLoop(dataRx, hD, prmLTEPDSCH, nVar);
end
% Demap received codeword(s)
[cwOut, ~] = LayerDemapper(yRec, prmLTEPDSCH);
if prmLTEPDSCH.Eqmode < 3
   % Demodulate
   demodOut = DemodulatorSoft(cwOut, prmLTEPDSCH.modType, mean(nVar));
else
   demodOut = cwOut;
end
% Descramble received codeword
rxCW = lteDescramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes - CB segmentation, turbo decoding, rate dematching
[decTbData1, ~,~] = lteTbChannelDecoding(nS, rxCW, Kplus1, C1, prmLTEDLSCH,
prmLTEPDSCH);
% Transport block CRC detection
[dataOut, ~] = CRCdetector(decTbData1);
end
```

6.7.8.1 发射端模型结构

下面的 MATLAB 测试脚本调用 MIMO 收发端函数 commlteMIMO。首先，调用初始化函数（commlteMIMO _ initialize）设置所有相关参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。然后在一个 While 循环内调用 MIMO 收发端函数（commlteMIMO _ SM3 _ step）进行子帧处理。最后，计算 BER 并调用可视化程序显示信道响应和均衡前后的调制星座图。

Algorithm

MATLAB script

```
% Script for MIMO LTE (mode 3)
%
% Single or Two codeword transmission
%
clear all
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, ...
   chEstOn, numCodeWords, snrdB, maxNumErrs, maxNumBits);
```

```
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl corrLvl chEstOn numCodeWords snrdB maxNumErrs
maxNumBits
%%
disp('Simulating the LTE Mode 3: Multiple Tx & Rx antrennas with Spatial
Multiplexing');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
hPBer = comm.ErrorRate;
snrdB=prmMdl.snrdB;
maxNumErrs=prmMdl.maxNumErrs;
maxNumBits=prmMdl.maxNumBits;
%% Simulation loop
tic;
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
  [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
     commlteMIMO_SM_Mode3_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH,
prmMdl);
   % Calculate bit errors
   Measures = step(hPBer, dataIn, dataOut);
    % Visualize constellations and spectrum
   if (visualsOn && prmLTEPDSCH.Eqmode~=3)
     zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
   end;
   % Update subframe number
   nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
end
disp(Measures);
toc;
```

6.7.8.2 验证收发端性能

通过执行 MIMO 收发器模型（commlteMIMO）的 MATLAB 测试脚本，我们可以通过观察各个信号评估系统性能。仿真涉及的参数总结在下面的 MATLAB 脚本（commlteMIMO_params）。这些参数设定与 6.7.1 节相同。参数可以反映双码字空分复用个 MMSE 均衡 MIMO 模式 3 的不同之处。本仿真处理一百万用户数据比特，AWGN 信道的 SNR 为 16dB，可视化输出。

Algorithm

MATLAB script

```
% PDSCH
txMode        = 3;   % Transmission mode one of {1, 2, 4}
numTx         = 2;   % Number of transmit antennas
numRx         = 2;   % Number of receive antennas
chanBW        = 4;       % [1,2,3,4,5,6] maps to [1.4, 3, 5, 10, 15, 20]MHz
contReg       = 1;       % {1,2,3} for >=10MHz, {2,3,4} for <10Mhz
modType       = 2;       % [1,2,3] maps to ['QPSK','16QAM','64QAM']
```

```
% DLSCH
cRate           = 1/3; % Rate matching target coding rate
maxIter         = 6;   % Maximum number of turbo decoding terations
fullDecode      = 0;   % Whether "full" or "early stopping" turbo decoding is performed
% Channel model
chanMdl         = 'frequency-selective-high-mobility';
% one of {'flat-low-mobility', 'flat-high-mobility','frequency-selective-low-mobility',
% 'frequency-selective-high-mobility', 'EPA 0Hz', 'EPA 5Hz', 'EVA 5Hz', 'EVA 70Hz'}
corrLvl         = 'Medium';
% Simulation parameters
Eqmode          = 2;   % Type of equalizer used [1,2,3] for ['ZF', 'MMSE','Sphere Decoder']
chEstOn         = 1;       % use channel estimation or ideal channel
snrdB           = 16;  % Signal to Noise Ratio in dB
maxNumErrs      = 1e6; % Maximum number of errors found before simulation stops
maxNumBits      = 1e6; % Maximum number of bits processed before simulation stops
visualsOn       = 1;     % Whether to visualize channel response and constellations
numCodeWords = 1; % Number of codewords in PDSCH
enPMIfback      = 0;   % Enable/Disable Precoder Matrix Indicator (PMI) feedback
cbIdx           = 1;     % Initialize PMI index
```

图 6.17 所示为双天线中每一个天线上一个子帧的用户数据均衡前（第一行）后（第二行）星座图。可以看到均衡器补偿了信道衰落的影响，使星座图更接近于 16QAM。

图 16.8 所示为双天线中每一个天线上一个子帧的用户数据频谱。图中可以看到发射信号以及接收信号均衡前后的频谱。均衡前（可看到频率选择性衰落）的接收信号在开环空分复用后（可看到更平坦频谱），更接近发射信号频谱。

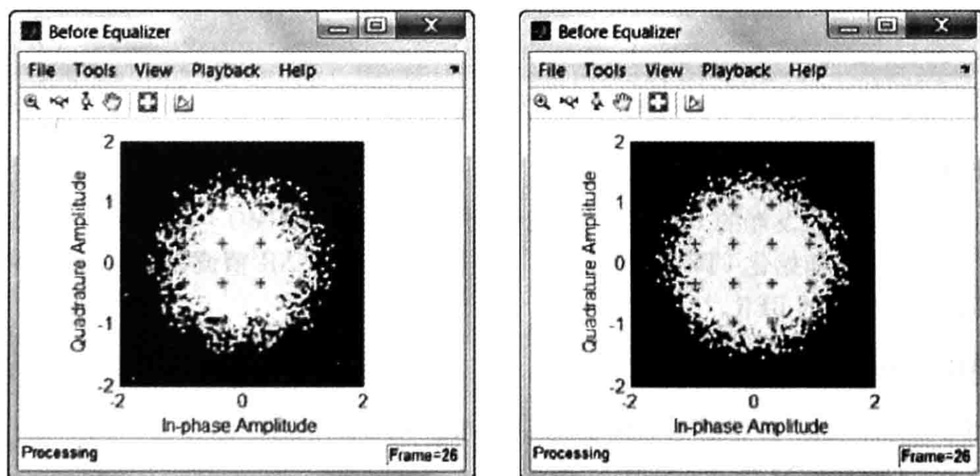


图 6.17　LTE 模型：MIMO 空分复用用户数据均衡前后的星座图

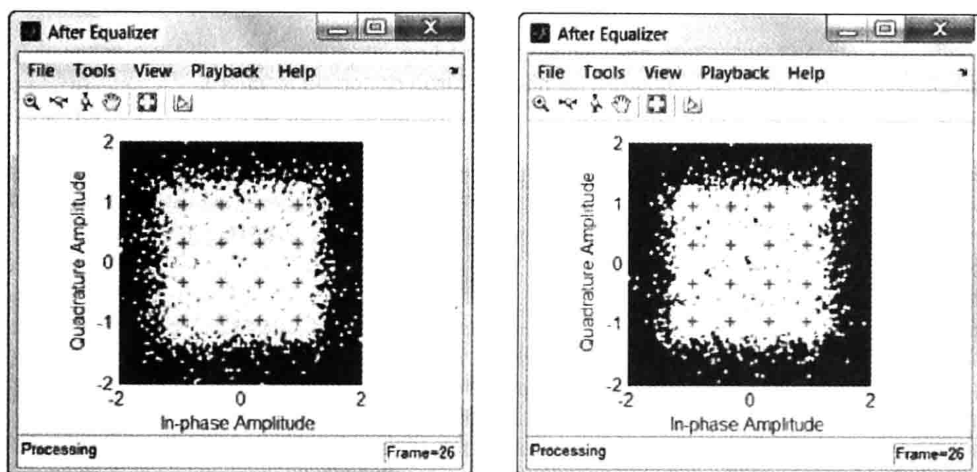


图 6.17 LTE 模型：MIMO 空分复用用户数据均衡前后的星座图（续）

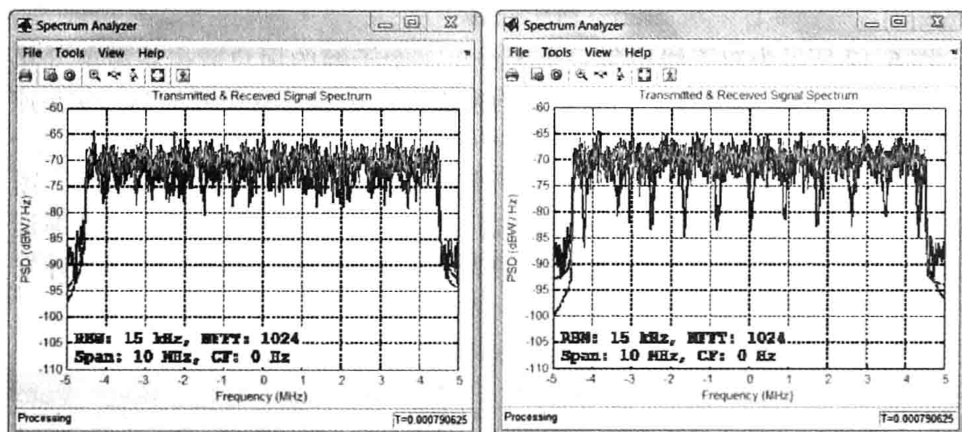


图 6.18 LTE MIMO 空分复用发射信号和均衡前后接收信号的频谱

6.7.8.3 BER 测量

为了验证收发端的 BER 性能，我们创建了 commlteMIMO _ test _ timing _ ber 的测试脚本，初始化 LTE 系统变量，并在循环中遍历 SNR 值调用 commlteMIMO _ fcn 计算相应的 BER。

Algorithm

MATLAB script: commlteMIMO_test_timing_ber

```
% Script for MIMO LTE (mode 4)
%
% Single codeword transmission only
%
clear all
```

```
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_params;
maxNumErrs=5e7;
maxNumBits=5e7;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl corrLvl chEstOn numCodeWords enPMIfback cbIdx snrdB
maxNumErrs maxNumBits
%%
disp('Simulating the LTE Mode 3: Multiple Tx & Rx antrennas with Spatial Multiplexing');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
%% Geerate code and setup parallelism
disp('Generating code for commlteMIMO_fcn.m ...');
arg1=coder.Constant(prmLTEPDSCH);
arg2=coder.Constant( prmLTEDLSCH);
arg3=coder.Constant(prmMdl);
codegen commlteMIMO_fcn -args {16, arg1, arg2, arg3} -report
disp('Done.');
parallel_setup;
%%
MaxIter=8;
snr_vector=getSnrVector(prmLTEPDSCH.modType, MaxIter);
ber_vector=zeros(size(snr_vector));
maxNumBits=prmMdl.maxNumBits;
tic;
parfor n=1:MaxIter
   fprintf(1,'Iteration %2d out of %2d:  Processing %10d bits. SNR = %3d\n', ...
      n, MaxIter, maxNumBits, snr_vector(n));
   [ber, ~] = commlteMIMO_fcn_mex(snr_vector(n), prmLTEPDSCH, prmLTEDLSCH,
prmMdl);
   ber_vector(n)=ber;
end;
toc;
semilogy(snr_vector, ber_vector);
title('BER - commlteMIMO SM');xlabel('SNR (dB)');ylabel('ber');grid;
```

图 6. 19 所示为收发端 BER 与 SNR 的关系，程序进行八次迭代，每次处理 5000 万比特用户数据。

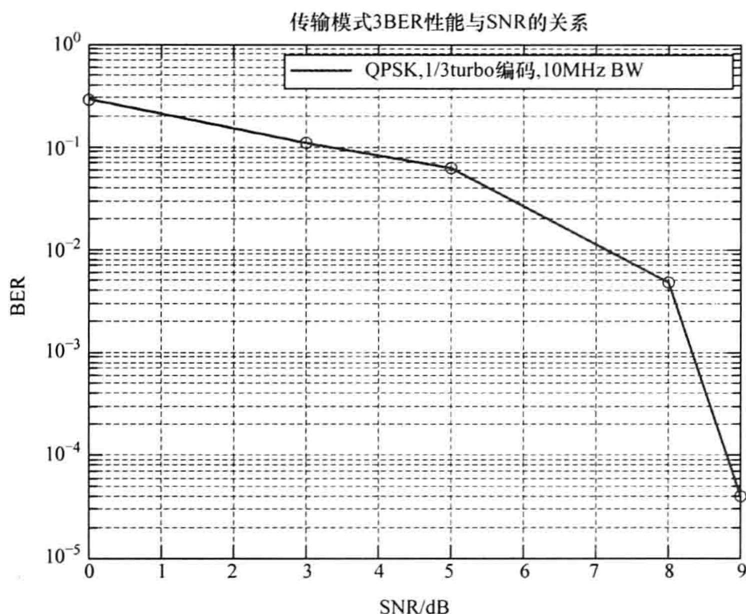


图 6.19 BER 结果：LTE 模式 3 空分复用单码字（2×2）MIMO 信道

6.8 本章小结

在本章中我们研究了 LTE 标准中多天线 MIMO 技术。MIMO 技术为 LTE 的核心部分。如下行链路传输中九种模式之间的差别，在于各自使用不同的 MIMO 技术特性。我们关注 LTE 中前四种 MIMO 算法并使用 MATLAB 建模。这些传输模式采用两种算法：发射分集（如 SFBC）和空分复用，以及是否使用延迟分集。发射分集技术提升链路质量和可靠性，单对提升数据速率和频带效率没有帮助。空分复用技术则大幅提升数据速率。

我们首先考察了 MIMO 多径衰落信道模型，随后考察 MIMO 传输方案中除发射分集和空分复用之外的基础组件。我们对前面章节介绍的 OFDM 基础组件进行升级，引入了多天线。我们将单天线数据的 2D 时 - 频分布扩展为 3D 时 - 频 - 空分布，并对一般 MIMO 算法包括资源元素映射、信道估计，和信道响应抽取进行算法升级。

随后，我们研究了发射分集与空分复用这两个特殊 MIMO 技术。LTE 标准中在发射端体现这两种技术的地方在层映射和预编码。我们同时考察了接收端处理，该过程反向了发射端处理，复原得到 3D 资源网格的最优估计。我们考察了三种 MIMO 接收器——ZF、MMSE，和 SD 算法——它们提供了给定采样时间多

天线子载波上发射数据的估计。

最后，我们统合所有功能组件，针对 LTE 标准传输模式 2、模式 3 和模式 4 建立 MATLAB 收发器模型。模式 2 基于发射分集，模式 3 使用开环空分复用，模式四为闭环空分复用。通过仿真，我们进行定量分析和 BER 性能评估。结果显示收发器可以有效抑制由多径衰落造成的码间串扰，并根据传输模式不同获得高数据速率。

参考文献

[1] Dahlman, E., Parkvall, S. and Sköld, J. (2011) *4G LTE/LTE-Advanced for Mobile Broadband*, Elsevier.
[2] Jafarkhani, H. (2005) *Space-Time Coding; Theory and Practice*, Cambridge University Press, Cambridge.
[3] 3GPP Evolved Universal Terrestrial Radio Access (E-UTRA); Base Station (BS) Radio Transmission and Reception, May 2011, TS 36.104.
[4] Scaglione, P., Stoica, S., Barbarossa, G. *et al.* (2002) Optimal designs for space-time linear precoders and decoders. *IEEE Transactions on Signal Processing*, 50, 5, 1051–1064.
[5] Browne, M. and Fitz, M. (2006) Singular value decomposition of correlated MIMO channels. IEEE Global Telecommunications Conference (GLOBECOM) 2006.
[6] Adhikari, S. (2011) Downlink transmission mode selection and switching algorithm for LTE. Proceedings of 3rd International Conference on Communication Systems and Networks (COMSNETS), January 2011.
[7] 3GPP (2011) Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation V10.0.0. TS 36.211, January 2011.

7 链路自适应

在前面各章我们研究了 LTE 标准中的调制、编码、绕码、信道建模、多载波，和多天线传输方案。我们详细考察了多个 MIMO 传输模式，和每种模式适用的最佳条件，以及系统可以达到的最高数据速率。我们并没有关注各个传输模式的瞬态情况或它们相互切换的机制。在本章中，我们将会对 LTE 标准的动态特性，以及通过什么方式决定各个参数在时域上瞬态变化的信道条件下优化频谱效率进行概览。

频谱效率是衡量移动通信系统的一个重要测量指标。在 LTE 标准中，有各种与 3G（第三代）标准相关的特征指标如平均值、小区边界，以及总频谱效率等[1]。频谱效率的定义为一个小区单位带宽（Hz）的数据速率平均值。该定义体现了设计移动系统需要进行的折中。即对一个给定的带宽，我们可以通过用高阶调制或高阶 MIMO 技术提高数据速率以提升频谱效率。而在噪声信道环境中，这些方法会增加误码率从而影响有效吞吐量。

为了稳定的获得所需的频谱效率，3G 和 4G 标准，包括 LTE，采用了可以根据信道条件动态改变系统参数的技术。该技术即信道感知调度或链路自适应。

链路自适应的基本思想是根据系统测量和监控的信道条件适配特定的传输参数。典型的动态自适应系统参数包括系统带宽、MIMO 传输模式、传输层数、预编码矩阵、调制和编码方案（MCS），和传输功率。通过适当的选择这些系统参数，我们可以更有效利用带宽资源，而不需要针对某一种最坏信道条件固定某个参数设定以期获得最优性能。

在本章中，我们将首先回顾移动接收端进行的各种测量以弄清信道条件和时间的关系。这些测量包括信道质量指示（Channel Quality Indicator，CQI）、预编码矩阵指示（Precoder Matrix Indicator，PMI），和秩指示（Rank Indicator，RI）测量。然后我们会讨论响应信道测量和改变多个系统参数以维持给定信道质量测定的技术。这些技术包括自适应 MCS、闭环空分复用模式自适应编码、基于秩估计的自适应 MIMO。最后，我们会对 PUCCH（物理下行链路控制信道）和 PD-CCH 物理下行链路信道进行一个简短的概览。这两个信道可使 UE（用户设备）和 eNodeB（演进形节点基站）之间进行信道测量和自适应调度通信。

7.1 系统模型

链路自适应是适应链路条件并依据实际信道质量改变系统参数的技术。LTE标准中的链路自适应帮助我们提高频谱效率。自适应技术带来的牺牲在链路感知调度器的计算复杂度方面。图 7.1 表示了和链路自适应有关的一般操作，包括上行链路和下行链路操作。

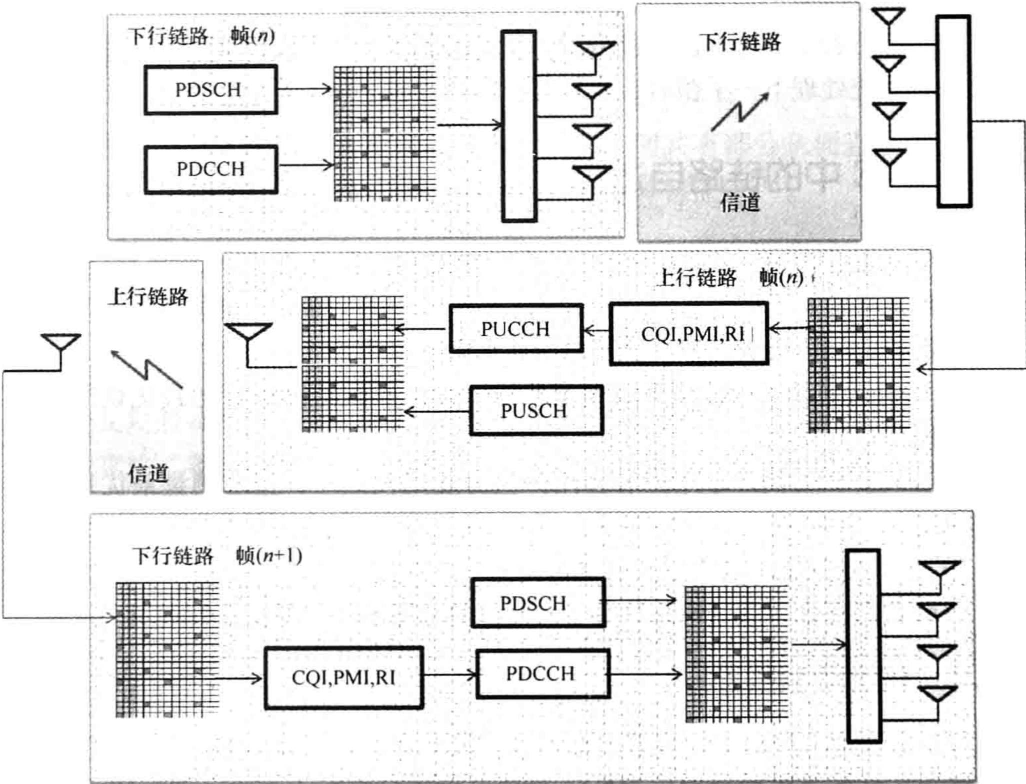


图 7.1 和链路自适应有关的下行链路和上行链路操作链

这一系列和链路自适应有关的操作可以如下总结：

1）在子帧（n）上，下行链路发射端由用户数据（PDSCH，物理下行链路公共信道）和下行链路控制，以及 DCI（PDCCH）构建资源网格，包含调度分配信息的 DCI 帮助移动接收端正确译码子帧信息。PDCCH 中的信息包括 MCS、预编码矩阵、秩信息，和 MIMO 模式。

2）在接收资源网格译码过程中，移动接收端进行关键一步——信道条件测量。在该处理中，移动接收端估计接收信道矩阵并进行多种信道质量测量。这些

测量包括 CQI、PMI，和 RI。

3）作为上行链路传输的一部分，UE 发射端在 PUCCH 中内嵌信道质量测量，并通过闭环反馈机制将其发送至基站（eNodeB）。

4）基站（eNodeB）接收端随后译码 PUCCH 信息得到信道测量结果。这一信息帮助系统调度器判断在下一帧是否适配各个系统参数，以作为下行链路信道质量的反馈结果。

5）在基站（eNodeB）下行链路发射端处理下一子帧（$n+1$）时，PDCCH 信息打包调度器基于信道条件的决策并发送至移动终端。这些信息包括更新的 MCS、预编码矩阵、秩信息，和 MIMO 模式，它们适配上一子帧（n）的实际信道质量。系统处理下一子帧时，重复这一反馈过程。

7.2 LTE 中的链路自适应

为了动态改变 MCS 以及进行更合适的 MIMO 处理，LTE 标准定义 UE 测量信道特征信息并将其反馈至基站（eNodeB），以帮助调度器判定和链路自适应。

移动接收端可生成三种反馈基站的信道状态报告：

1）CQI，测量下行链路无线信道质量，确定匹配链路质量的最优调制星座集和编码率；

2）PMI，LTE 闭环单用户和多用户空分复用模式下，指示测量最优预编码矩阵设定；

3）RI，空分复用模式下发射端可用的传输层数。

下面我们将会详细讨论每种报告并对测量计算的各个算法进行通览。

7.2.1 信道质量估计

CQI 报告给出移动无线信道质量测量。该测量根据 MCS 得到一个最优推荐。该推荐对应的信道传输块误码率低于 10%，并以此得到测量值。高 CQI 测量值代表了高调制阶数和高码率。CQI 报告根据分配 MCS 的不同分为两种类型：宽带 CQI 报告对全系统带宽分配一个 MCS 值；子带 CQI 报告对相邻的不同资源区块分配多个 MCS 值。

文献［2-6］中有多种方法优化 MCS 选择。大多数选择最优 MCS 的技术基于后检 SINR（信号干扰噪声比）测量。该测量选择包误码率（PER）小于给定目标值的 MCS。此过程可以使系统避免反复重发。最终通过码书查询表[2]量化 SINR 值可以的得到最优 MCS 推荐。

7.2.2 预编码矩阵估计

PMI 报告给出下行链路传输闭环空分复用的预编码码书索引。与 CQI 报告相同，PMI 报告也可以分为单个宽带值和多个子带值。在文献中有各种 PMI 选择方法。典型的选择标准总结在参考［7］中。这些标准根据优化过的预编码矩阵而各不相同。优化选择方法包括最小奇值和最小均方误差（MSE）或容量法。

7.2.3 秩估计

秩估计（RI）对空分复用的传输层数或独立数据流数进行测量。文献中有各种估计信道矩阵秩的方法。文献［8］中的一些方法基于后检 SINR，与选择 MCS 相同。其他方法为扩大发射信号与后检信号间共有部分从而直接扩大容量的方法[11]。另有一个略简单的方法是使用信道矩阵特征值[7]。

7.3 MATLAB 实例

在本节中，我们回顾发射端生成信道状态报告的 MATLAB 算法。因标准中没有定义发射端操作，针对单用户，我们选择可接受的计算复杂度和稳定性为标准选择算法。我们将这些算法作为用 MATLAB 实现信道状态测量和链路自适应模型的基石和基本框架。

7.3.1 CQI 估计

本节中两个 MATLAB 函数基于 MIMO 接收器输出和发射信号的 SINR，实现信道质量估计（CQI）测量。CQI 估计分为如下两步：

1）SINR 估计：由接收端和 MIMO 接收器输出的译码比特计算 SINR 估计；

2）频谱效率查询：调制映射上一步得到的 SINR 到码率频谱效率测量值。频谱效率测量值由单位符号调制比特数与码率相乘得到。对每个 SINR 测量值，可通过查询表得到对应的调制方案和码率。

7.3.1.1 SINR 估计

定义 G 为最优均衡器。它将接收信号 $Y(n)$ 变换为均衡信号 $\hat{X}(n)$，作为发射信号 $X(n)$的最优线性估计。则误差信号 $e(n)$可表示为

$$e(n) = \hat{X}(n) - X(n) = GY(n) - X(n) \tag{7.1}$$

为了进行 CQI 估计，我们对 SINR 测量值进行一个简单的近似，定义其为发射信号功率 σ_x^2 和误差信号功率 σ_e^2 的比。

$$SINR = 10\log_{10}\left(\frac{\sigma_x^2}{\sigma_e^2}\right) \tag{7.2}$$

下面的函数（CQIselection. m）为计算 SINR 测量值，输入为接收端译码比特、后检 MIMO 接收器输出（equalized）、当前帧数（nS），和 PDSCH 以及 DLSCH（下行链路公共信道处理）参数结构体（prmLTEDLSCH，prmLTEPDSCH）。函数输出为 SINR 估计（sinr）。注意我们需要发射信号 $X(n)$ 的最优估计来计算 SINR，它由函数变量 modOut 得到。函数通过处理译码比特（输入变量 bit）求出 modOut。接收端输出比特即每子帧输入比特的最优估计。我们在本书中一直用该信号计算和考察系统的比特误码率。函数通过在开始调用的几个发射端函数得到调制信号的最优估计。这些发射端函数操作包括 CRC（循环冗余校验）添加、信道编码、绕码，和调制。最后，函数按照式（7.2）计算 SINR 估计。

Algorithm

MATLAB function

```
function sinr=CQIselection(bits, equalized, nS, prmLTEDLSCH, prmLTEPDSCH)
%#codegen
tbCrcOut1 =CRCgenerator(bits);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
data = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH, prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
error=modOut-equalized;
sinr=10*log10(var(modOut)./var(error));
```

7.3.1.2 频谱效率查询

下面的函数实现 SINR（输入变量 sinr）映射，分别得到对应的调制方案和码率（输出变量 Ms 和 Cr）。通过使用一个 4bit（16 间隔）标量量化器，我们首先将 SINR 值映射到 CQI 索引。DSP 系统工具箱中的系统对象 dsp. ScalarQuantizerEncoder 可以用来完成这一操作。阈值对应了标量量化器的分界点。因量化器定义为广义边界（输入值为 – inf 到 + inf），我们只需要 15 个阈值将实轴分割为 16 个区间，对应四个 CQI 比特。通过一个简单的查询表，这些阈值将 SINR 值映射到相应的频谱效率上。

Algorithm

MATLAB function

```
function [Ms, Cr]=CQI2indexMCS(sinr)
%#codegen
% Table of SINR threshold values, 15 boundary points for an unbounded quantizer
thresh=[-6.7,-4.7,-2.3,0.2,2.4,4.3,5.9,8.1,10.3,11.7,14.1,16.3,18.7,21,22.7];
% Table of coding rate (16 value)
Map2CodingRate=[0.076, 0.076, 0.117, 0.188, 0.301, 0.438, 0.588, 0.369, 0.479,...
0.602, 0.455, 0.554, 0.650, 0.754, 0.853, 0.926];
% Table of modulation type (1=QPSK, 2=QAM16, 3=QAM64)
Map2Modulator=[1*ones(7,1);2*ones(3,1);3*ones(6,1)];
persistent hQ
if isempty(hQ)
   hQ=dsp.ScalarQuantizerEncoder(...
      'Partitioning', 'Unbounded',...
      'BoundaryPoints', thresh,...
      'OutputIndexDataType','uint8');
end;
indexCQI=step(hQ, sinr);
index1=indexCQI+1;                  % 1-based indexing
% Map CQI index to modulation type
Ms = Map2Modulator (index1);
% Map CQI index to coding rate
Cr = Map2CodingRate (index1);
if Cr < 1/3, Cr=1/3;end;
```

我们通过 CQI 索引在查询表中找相对应的调制方案和码率值。对于表中前 7 个 CQI 索引（序号 0～6），对应每符号 2bit 调制率的 QPSK（正交相移键控）。下面 3 个 CQI 索引（7、8，9）对应每符号 4bit 调制率的 16QAM（正交幅度调制）。最后 6 个 CQI 索引（10～15）对应每符号 6bit 调制率的 64QAM。一般的，CQI 索引 0 不在调制映射表内。它用来标记溢出消息。为了简要起见，我们的 MATLAB 函数只处理 QPSK 对应的 CQI 索引。注意表内另有与调制方案相对应的码率（Cr）和频谱效率的 16 个值，它们由 LTE 标准文档[10]定义。这个查询表见表 7.1。

表 7.1　SINR 测量对应调制方案和码率的查询表

CQI 索引	调制	码率	频谱效率/（bit/s/Hz）	SINR 估计/dB
1	QPSK	0.0762	0.1523	-6.7
2	QPSK	0.1172	0.2344	-4.7
3	QPSK	0.1885	0.3770	-2.3
4	QPSK	0.3008	0.6016	0.2

（续）

CQI 索引	调制	码率	频谱效率/（bit/s/Hz）	SINR 估计/dB
5	QPSK	0.4385	0.8770	2.4
6	QPSK	0.5879	1.1758	4.3
7	16QAM	0.3691	1.4766	5.9
8	16QAM	0.4785	1.9141	8.1
9	16QAM	0.6016	2.4063	10.3
10	64QAM	0.4551	2.7305	11.7
11	64QAM	0.5537	3.3223	14.1
12	64QAM	0.6504	3.9023	16.3
13	64QAM	0.7539	4.5234	18.7
14	64QAM	0.8525	5.1152	21.0
15	64QAM	0.9258	5.5547	22.7

7.3.2 PMI 估计

本节的 MATLAB 函数实现 PMI 码书索引选择。它用最小均方误差（MMSE）计算每子帧的 PMI 码书索引（cbIdx）。函数输入为接收端 3D 信道矩阵（h）、表示是否使用 PMI 闭环反馈的布尔值（enPMIfback）、发射天线数（numTx）、层数（或称为足秩可用发射天线数）（numLayers），和噪声方差（nVar）。假如不使用 PMI 闭环反馈，函数输出的码书索引恒定为 1。其他情况下，函数以一个最小距离对每个子帧赋一个码书索引值。

Algorithm

MATLAB function

```
function cbIdx = PMICbSelect(h, enPMIfback, numTx, numLayers, nVar)
%#codegen
% Codebook selection using minimum MSE criterion
if (enPMIfback)
   if (numTx == 2)
      cbLen = 2; % Only indices 1 and 2 are used for 2-layer closed-loop Spatial MUX
      MSEcb = zeros(cbLen, 1);
      for cbIdx = 1:cbLen
         Wn = PrecoderMatrix(cbIdx, numTx, numLayers);
         MSEcb(cbIdx) = Sinr_MMSE(h, nVar, Wn);
      end
      [~, cbIdx] = min(MSEcb); % 0-based, note 0 and 3 are not used
   else % for numTx=4
      cbLen = 2^numLayers;
      MSEcb = zeros(cbLen, 1);
      for cbIdx = 1:cbLen
         Wn = PrecoderMatrix(cbIdx-1, numTx, numLayers);
         MSEcb(cbIdx) = Sinr_MMSE(h, nVar, Wn);
```

```
end
[~, cbIdx] = min(MSEcb);  % 1-based
cbIdx = cbIdx-1;          % 0-based
   end
else
   cbIdx = 1;
end
end
% Helper function
function out = Sinr_MMSE(chEst, nVar, Wn)
%#codegen
% post-detection SNR computation
% Based on received channel estimates
% Per layer noise variance
% Precoder matrix
% Uses the MMSE detector.
% Get params
persistent Gmean
if isempty(Gmean), Gmean=dsp.Mean('RunningMean', true);end
noisFac = diag(nVar);
numData = size(chEst, 1);
numLayers = size(Wn,1);
F = inv(Wn);
%% MMSE receiver
for n = 1:numData
   h = chEst(n, :, :);                      % numTx x numRx
   h = reshape(h(:), numLayers, numLayers).'; % numRx x numTx
   Ht= inv((F'*(h'*h)*F) + noisFac);
   % Post-detection SINR
   g=real((1./(diag(Ht).*(nVar.')))-1);
   Gamma=step(Gmean,g);
end
out=mean(Gamma);
reset(Gmean);
end
```

首先测量后检 MIMO 接收器估计和发射调制器的 MSE。该测量从公式上包含 MIMO 信道矩阵和预编码矩阵。每个预编码矩阵通过循环遍历所有 PMI 码书进行，即进行全码书搜索。该测量计算涉及码书内每个索引以及 3D 信道矩阵每个时间采样（第一阶）。最终得到最小 MSE 测量的码书索引值。注意对四天线传输，我们遍历搜索 16 个码书索引，而对两天线情况，我们遍历 2bit 表示的码书子集。

7.3.3 RI 估计

本节的 MATLAB 函数实现基于信道矩阵状态数的 RI 估计算法。状态数定义为信道矩阵最大和最小特征值的比。这个比值可以很好表征矩阵求反的精确度和系统线性方程是否可解。状态数的值接近 1 代表信道矩阵状态很好。而大比值代

表信道矩阵不佳，其所代表系统线性方程不可解。函数调用了 MATLAB 函数 cond 得到信道矩阵状态数的数值解。

Algorithm

MATLAB function

```
function y = RIestimate(Q)
%#codegen
y=cond(Q);   % Condition number of a matrix
```

接收端必须对每个 2D 信道矩阵的进行秩估计运算。这个 2D 矩阵为 3D 信道矩阵在每个采样时间（第一阶）的样本。MIMO 接收器的 for 循环是执行秩估计运算的最好位置，我们循环遍历 3D 信道矩阵的第一阶，采样计算后检 MIMO 接收器输出样本。因此，我们需要更新 MIMO 接收器函数添加进秩估计，并将其结果作为附加的输出。

7.3.3.1 MIMO 接收端函数中的 RI 计算

下面的 MIMO 接收端函数在循环中添加了秩估计。在第 6 章中，我们基于迫零（ZF）、MMSE，和球型译码器（SD）算法开发了三种不同的 MIMO 接收器。

下面的函数在 ZF MIMO 接收器循环中一个样本一个样本的计算 RI 估计。秩估计输出（ri）为 MIMO 信道状态数构成的列向量，每一个后检接收器输出（y）对应一个值。

Algorithm

MATLAB function

```
function [y, ri] = MIMOReceiver_ZF(in, chEst, Wn)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the ZF detector.
% Get params
numData = size(in, 1);
y = complex(zeros(size(in)));
ri=zeros(numData,1);
iWn = inv(Wn);
%% ZF receiver
for n = 1:numData
    h = squeeze(chEst(n, :, :)); % numTx x numRx
    h = h.';                     % numRx x numTx
   ri(n) = RIestimate(h);
    Q = inv(h);
    x = Q * in(n, :).';%#ok
    tmp = iWn * x; %#ok
    y(n, :) = tmp.';
end
```

与上一个函数相似，下面的函数在 MMSE MIMO 接收器内一个样本一个样本计算 RI 估计输出（ri）。秩估计输出（ri）为 MIMO 信道状态数构成的列向量，每一个后检接收器输出（y）对应一个值。

Algorithm

MATLAB function

```
function [y, ri] = MIMOReceiver_MMSE(in, chEst, nVar, Wn)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the MMSE detector.
% Get params
numLayers = size(Wn,1);
% noisFac = numLayers*diag(nVar);
noisFac = diag(nVar);
numData = size(in, 1);
y = complex(zeros(size(in)));
ri=zeros(numData,1);
iWn = inv(Wn);
%% MMSE receiver
for n = 1:numData
    h = chEst(n, :, :);                    % numTx x numRx
    h = reshape(h(:), numLayers, numLayers).'; % numRx x numTx
    ri(n) = RIestimate(h);
    Q = (h'*h + noisFac)\h';
    x = Q * in(n, :).';
    tmp = iWn * x; %#ok
    y(n, :) = tmp.';
end
```

最后一个函数在球型译码器 MIMO 接收器内一个样本一个样本计算 RI 估计输出（ri）。秩估计输出（ri）为 MIMO 信道状态数构成的列向量，每一个后检接收器输出（y）对应一个值。

Algorithm

MATLAB function

```
function [y, ri, bittable] = MIMOReceiver_SphereDecoder(in, chEst, prmLTE, nVar, Wn)
%#codegen
% MIMO Receiver:
%   Based on received channel estimates, process the data elements
%   to equalize the MIMO channel. Uses the Sphere detector.
% Soft-Sphere Decoder
symMap=prmLTE.SymbolMap;
numBits=prmLTE.Qm;
constell=prmLTE.Constellation;
```

```
bittable = de2bi(symMap, numBits, 'left-msb');
iWn=Wn.';
nVar1=(-1/mean(nVar));
ri=zeros(numData,1);
persistent SphereDec
if isempty(SphereDec)
   % Soft-Sphere Decoder
   SphereDec = comm.SphereDecoder('Constellation', constell,...
      'BitTable', bittable, 'DecisionType', 'Soft');
end
% SSD receiver
temp = complex(zeros(size(chEst)));
% Account for precoding
for n = 1:size(chEst,1)
   h= squeeze(chEst(n, :, :));
   temp(n, :, :) = iWn * h;
    ri(n) = RIestimate(h);
end
hD = temp;
y = nVar1 * step(SphereDec, in, hD);
```

下面的函数根据秩估计平均值函数设定阈值更新传输模式。

Algorithm

MATLAB function

```
function y=RIselection(ri, threshold)
Ri=mean(ri);
% RI estimation
if Ri > threshold, y = 4; else y=2; end
```

7.4 子帧间的链路自适应

在 7.5 ~ 7.8 节，我们会看到利用信道状态信息（CSI：CQI、PMI，和 RI 估计）在子帧间适配各个收发端参数的各种方法。在本节中我们特别关注一些简单的调度实例。这些算法用 MATLAB 构建了一个实现自适应算法的框架。不过，在现实中实现调度决策的算法包括更多参数，包括 CSI、服务质量，以及正在传输的数据类型。

在下面的链路自适应实例中，我们将信道估计测量结果直接反馈到下一子帧的调度系统参数。我们针对下一子帧的所有资源区块使用同一个给定的自适应设定，即宽带自适应。另外，LTE 标准允许每个子帧不同的资源区块有不同的自适应设定，即子带自适应。为了简化算法的复杂度，本章不讨论子带自适应。

下面，我们讲解应用于单码字闭环空分复用（LTE 传输模式 4 单码字模型）的 4 种自适应技术。我们将首先进行自适应调制，随后讲解使用 CQI 进行自适应调制和编码的机制。然后我们会统和自适应调制和基于 PMI 测量进行的预编码选择。最后，我们统和前面所有自适应技术，并加入基于 RI 测量的自适应层映射。

7.4.1 收发端模型结构

下面的 MATLAB 测试脚本调用 MIMO 收发端函数。首先，调用初始化函数（commlteMIMO _ initialize）设置所有相关参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。然后在一个 While 循环内调用 MIMO 收发端函数进行子帧处理。

Algorithm

MATLAB script

```
% Script for MIMO LTE (mode 4)
%
% Single codeword transmission
%
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
   chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, Doppler, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl Doppler corrLvl chEstOn numCodeWords enPMIfback cbIdx snrdB
maxNumErrs maxNumBits
%%
disp('Simulating the LTE Mode 4: Multiple Tx & Rx antennas with Spatial Multiplexing');
zReport_data_rate(prmLTEPDSCH, prmLTEDLSCH);
hPBer = comm.ErrorRate;
snrdB=prmMdl.snrdB;
maxNumErrs=prmMdl.maxNumErrs;
maxNumBits=prmMdl.maxNumBits;
%% Simulation loop
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (Measures(3) < maxNumBits)
   % Insert one subframe step processing
   %% including adaptations here
end
BER=Measures(1);
BITS=Measures(3);
```

注意，我们在 While 循环内只加入了注释占位符。在脚本中，在注释“In-

sert one subframe step processing including adaptation here,” 的地方，我们将添加三种自适应技术的代码。

7.4.2 更新收发端参数结构体

为实现自适应机制，下面的函数更新三个参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。它的输入为初始化参数结构体（输入变量声明 p1，p2，和 p3）和附加输入变量声明。函数输出更新的参数结构体。

如果除了参数结构体（p1，p2，和 p3）之外还有一个的输入变量，则我们通过设置（modType）参数更新调制方案。如果有两个额外的输入，则首先更新调制方案然后更新码率（cRate）。这两个额外输入表示仅用 CQI 测量进行自适应调制和编码。如在 modType 和 cRate 之外还有第三个额外输入（cbIdx），则表征 PMI 码书索引。最后还有一个附加输入（txMode），表征我们使用空分复用模式（txMode = 4）还是发射分集模式（txMode = 2）。

Algorithm

MATLAB function

```
function [p1, p2, p3] = commlteMIMO_update(p1,p2, p3, varargin)
switch nargin
   case 1, modType=varargin{1}; cRate=p2.cRate; cbIdx=p3.cbIdx; txMode=p1.txMode;
   case 2, modType=varargin{1}; cRate=varargin{2}; cbIdx=p3.cbIdx; txMode=p1.txMode;
   case 3, modType=varargin{1}; cRate=varargin{2}; cbIdx=varargin{3};
txMode=p1.txMode;
   case 4, modType=varargin{1}; cRate=varargin{2}; cbIdx=varargin{3};
txMode=varargin{4};
   otherwise
      error('commlteMIMO_update has 1 to 4 arguments!');
end
% Update PDSCH parameters
tmp = prmsPDSCH(txMode, p1.chanBW, p1.contReg, modType,p1.numTx, p1.numRx, ...
   p1.numCodeWords,p1.Eqmode);
p1=tmp;
[SymbolMap, Constellation]=ModulatorDetail(p1.modType);
p1.SymbolMap=SymbolMap;
p1.Constellation=Constellation;
% Update DLSCH parameters
p2 = prmsDLSCH(cRate, p2.maxIter, p2.fullDecode, p1);
% Update channel model parameters
tmp = prmsMdl(txMode, p1.chanSRate,  p3.chanMdl, p3.Doppler, p1.numTx, p1.numRx, ...
   p3.corrLvl, p3.chEstOn, p3.enPMIfback, cbIdx, p3.snrdB, p3.maxNumErrs,
p3.maxNumBits);
p3=tmp;
```

7.5　自适应调制

在本节，我们利用 CQI 信道状态报告对收发端调制方案进行适配。我们进行宽带调制选择，即对全资源区块任意给定子帧用相同方案调制，在子帧之间改变调制方案。

为了理解设计折中，我们需要比较不同的适应性调制方法。我们有三种算法可以实现不同的自适应情况：

1）基准（无自适应）；

2）随机变更调制方案；

3）CQI 信道估计自适应调制方案。

下面我们通过在 While 循环中分别添加相应的代码考察三种情况。

7.5.1　无自适应

下面的 MATLAB 代码为 While 循环体内的部分。在不进行任何链路自适应情况下，While 循环体内只包括五个操作：

1）调用收发端函数处理一个数据子帧；

2）报告稳态平均和瞬态数据速率，以及平均编码率和调制率（每字符调制比特数）；

3）测量 BER（比特误码率）；

4）可视化后检接收信号和发射接收的 OFDM（正交频分复用）信号；

5）更新子帧数。

Algorithm

MATLAB script segment

```
%% One subframe step processing
[dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
    commlteMIMO_SM_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
 %% Report average data rates
ADR=zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
%% Calculate  bit errors
Measures = step(hPBer, dataIn, dataOut);
%% Visualize results
if (visualsOn && prmLTEPDSCH.Eqmode~=3)
    zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
end;
% Update subframe number
nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
%% No adaptations here
```

7.5.2　随机变更调制方案

下面的 MATLAB 代码为 While 循环体内的部分。在进行与无自适应情况相同的处理之外，While 循环体还包括两个操作：

1）为调制类型参数（modType）随机分配整型数 1、2，3，它们分别对应 QPSK、16QAM，和 64QAM；

2）调用 commlteMIMO _ updata 函数，根据新的调制类型更新 LTEPDSCH 和 LTEDLSCH 的所有参数。

Algorithm

MATLAB script segment

```
%% One subframe step processing
[dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
   commlteMIMO_SM_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
 %% Report average data rates
ADR=zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
%% Calculate  bit errors
Measures = step(hPBer, dataIn, dataOut);
%% Visualize results
if (visualsOn && prmLTEPDSCH.Eqmode~=3)
   zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
end;
% Change of modulation scheme randomly
modType=randi([1 3],1,1);
[prmLTEPDSCH, prmLTEDLSCH] = commlteMIMO_update( prmLTEPDSCH, prmLT-
EDLSCH, modType);
% Update subframe number
nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
%% No adaptations here
```

7.5.3　基于 CQI 的自适应

下面的 MATLAB 代码为 While 循环体内的部分。在进行与无自适应情况相同的处理之外，While 循环体还包括四个自适应操作：

1）CQI 测量报告，调用 CQIselection 函数计算 SINR 测量值；

2）通过调用 CQI2indexMCS 函数找到 SINR 对应的调制类型选择新的调制方案；

3）根据新的调制类型，调用 commlteMIMO _ update 函数更新 LTEPDSCH，LTEDLSCH，以及 prmMdl 参数；

4）调用 zVisSinr 函数描 24 个 SINR 测量的过去值，可视化信道质量随时间

的变化关系。

Algorithm

MATLAB script segment

```
%% One subframe step processing
[dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
    commlteMIMO_SM_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
%% Report average data rates
ADR=zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
%% CQI feedback
sinr=CQIselection(dataOut, yRec,  nS, prmLTEDLSCH, prmLTEPDSCH);
indexMCS=CQI2indexMCS(sinr);
%% Calculate  bit errors
Measures = step(hPBer, dataIn, dataOut);
%% Visualize results
if (visualsOn && prmLTEPDSCH.Eqmode~=3)
    zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
    zVisSinr(sinr);
end;
% Update subframe number
nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
% Adaptive change of modulation
modType=indexMCS;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_update
( prmLTEPDSCH, prmLTEDLSCH, prmMdl, modType);
```

7.5.4 收发端性能验证

仿真所用的所有参数总结在下面的 commlteMIMO _ params 脚本中。在仿真过程中，除了调制类型由变量 modType 之外，其他参数为常量。

Algorithm

MATLAB script

```
% PDSCH
txMode       = 4;  % Transmission mode one of {1, 2, 4}
numTx        = 2;   % Number of transmit antennas
numRx        = 2;   % Number of receive antennas
chanBW       = 6;        % [1,2,3,4,5,6] maps to [1.4, 3, 5, 10, 15, 20]MHz
contReg      = 1;        % {1,2,3} for >=10MHz, {2,3,4} for <10Mhz
modType      = 3;        % [1,2,3] maps to ['QPSK','16QAM','64QAM']
% DLSCH
cRate        = 1/3; % Rate matching target coding rate
maxIter      = 6;    % Maximum number of turbo decoding iterations
fullDecode   = 0;   % Whether "full" or "early stopping" turbo decoding is performed
```

```
% Channel
chanMdl        =  'frequency-selective'; % Channel model
Doppler        = 70;                     % Average Doppler shift
% one of {'flat-low-mobility', 'flat-high-mobility','frequency-selective-low-mobility',
% 'frequency-selective-high-mobility', 'EPA 0Hz', 'EPA 5Hz', 'EVA 5Hz', 'EVA 70Hz'}
corrLvl        = 'Medium';
% Simulation parameters
Eqmode         = 2;    % Type of equalizer used [1,2,3] for ['ZF', 'MMSE','Sphere Decoder']
chEstOn        = 1;       % use channel estimation or ideal channel
snrdB          = 20;   % Signal to Noise Ratio in dB
maxNumErrs     = 2e7; % Maximum number of errors found before simulation stops
maxNumBits     = 2e7;  % Maximum number of bits processed before simulation stops
visualsOn      = 0;     % Whether to visualize channel response and constellations
numCodeWords   = 1; % Number of codewords in PDSCH
enPMIfback     = 0;    % Enable/Disable Precoder Matrix Indicator (PMI) feedback
cbIdx          = 1;     % Initialize PMI index
```

函数 zReport _ data _ rate _ average 求平均和瞬态数据速率、编码率，和调制率。平均值由 DSP 系统工具包的 dsp. Mean 系统对象计算。瞬态数据速率由一个帧内 10 个子帧的总输入比特之和乘以 100 帧/s 的常系数得到。

Algorithm

MATLAB function

```
function t = zReport_data_rate_average(p2, p1)
persistent Rmean Rmod Rcod
if isempty(Rmean), Rmean=dsp.Mean('RunningMean', true);end
if isempty(Rmod), Rmod=dsp.Mean('RunningMean', true);end
if isempty(Rcod), Rcod=dsp.Mean('RunningMean', true);end
y=(1/10.0e-3)*(p1.TBLenVec(1)+p1.TBLenVec(2)+8*p1.TBLenVec(3));
z=y/1e6;
t=step(Rmean,z);
mm=step(Rmod,2*p2.modType);
cc=step(Rcod,p1.cRate);
Mod={'QPSK','16QAM','64QAM'};
fprintf(1,'Modulation                   = %s\n',Mod{p2.modType});
fprintf(1,'Instantaneous Data rate      = %.2f Mbps\n',z);
fprintf(1,'Average Data rate            = %.2f Mbps\n',t);
fprintf(1,'Instantaneous Modulation rate = %4.2f\n',2*p2.modType);
fprintf(1,'Average Modulation rate      = %4.2f\n',mm);
fprintf(1,'Instantaneous Coding rate    = %.4f\n',p1.cRate);
fprintf(1,'Average Coding rate          = %.4f \n\n',cc);
end
```

通过执行 MIMO 接收器自适应调制模型的测试脚本，我们可以观察评估系统性能。图 7. 2 显示，调制方案的改变影响瞬态数据速率并最终影响平均数据速率。

```
Modulation                        = 64QAM
Instantanous Data rate            = 30.58 Mbps
Average Data rate                 = 30.58 Mbps
Instantanous Modulation rate = 6.00
Average Modulation rate           = 6.00
Instantanous Coding rate          = 0.3333
Average Coding rate               = 0.3333

Modulation                        = QPSK
Instantanous Data rate            = 10.30 Mbps
Average Data rate                 = 20.44 Mbps
Instantanous Modulation rate = 2.00
Average Modulation rate           = 4.00
Instantanous Coding rate          = 0.3333
Average Coding rate               = 0.3333

Modulation                        = 16QAM
Instantanous Data rate            = 19.85 Mbps
Average Data rate                 = 20.24 Mbps
Instantanous Modulation rate = 4.00
Average Modulation rate           = 4.00
Instantanous Coding rate          = 0.3333
Average Coding rate               = 0.3333
```

图 7.2 链路自适应：子帧的数据速率、调制，和编码模式

（注：图中的“bps”即为“bit/s”）

7.5.5 结论

针对每种自适应情况，我们分别处理 2000 万比特以计算 BER。其结果总结在表 7.2 中。如我们预期，自适应调制可以很好适配信道质量变化。在无自适应情况下，当我们用高调制率的调制方式时，如 64QAM，我们在得到高数据速率的同时牺牲了 BER 性能。当我们用低调制率的调制方式时，如 QPSK，我们获得更低的 BER 但牺牲了数据速率。在随机选择调制方案情况下，由于调制方式与信道质量无关，数据速率和 BER 皆比无自适应情况略好。

表 7.2 自适应调制：不同情况下 BER、数据速率，和调制率

调制类型	平均数据速率/(Mbit/s)	调制率	码率	比特误码率
QPSK - 无自适应	20.61	2	0.3333	$1.2e^{-06}$
16QAM - 无自适应	39.23	4	0.3333	$1.4e^{-06}$
64QAM - 无自适应	61.66	6	0.3333	0.0033
随机选择	40.75	2 或 4 或 6	0.3333	0.0014
自适应调制	52.61	2 或 4 或 6	0.3333	0.0009

当依据信道质量选择调制方案时，我们在低或高信道质量情况下都得到最好

的折中。在高信道质量情况下，我们选择高调制率。即使我们的调制方案有较小的最小星座点间距，因信道质量非常好，子帧的误码概率很低，故我们可以获得最高数据速率且很少出现误码。在低信道质量情况下，我们选择较低调制率。较低调制率代表了较大的星座点间距，从而减小了误码发生概率。这样，我们在可接受范围内减小了数据速率并维持了通信质量。因此，自适应调制的平均 BER（0.0009）低于随机选择调制（0.0016），而数据速率（52.61Mbit/s）则高于随机选择调制（40.75Mbit/s）。我们通过考察基于信道质量的自适应，了解了较高数据速率与较低误码率的设计折中，并得到了一个最优的结果。

7.6 自适应调制与编码率

本节中，我们在收发端各子帧用 CQI 信道状态报告自适应改变调制方案和编码率，并和其他两种算法：

1）基准（无自适应）；

2）随机改变调制类型和码率进行对比。

在无自适应情况下，我们考察三种 LTE 调制方案。其码率等于使用 CQI 自适应技术的平均码率。在随机适配情况下，我们随机选择一种 LTE 调制方案，并从 CQI 自适应技术的码率值范围内随机选择一个值作为随机方案的码率。

7.6.1 无自适应

下面的 MATLAB 代码为 While 循环体内的部分。由于不使用自适应，这部分 MATLAB 代码与 7.5 节相同。

Algorithm

MATLAB script segment

```
%% One subframe step processing
[dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
   commlteMIMO_SM_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
 %% Report average data rates
ADR=zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
%% Calculate  bit errors
Measures = step(hPBer, dataIn, dataOut);
%% Visualize results
if (visualsOn && prmLTEPDSCH.Eqmode~=3)
   zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
end;
% Update subframe number
nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
%% No adaptations here
```

7.6.2 随机变更调制方案

下面的 MATLAB 代码为 While 循环体内的部分。在进行与无自适应情况相同的处理之外，While 循环体还包括两个操作：

1）为调制类型参数（modType）随机分配整型数 1、2，3，它们分别对应 QPSK、16QAM，和 64QAM；

2）随机从 1/3 到 0.95 之间的范围内选择一个值作为码率（cRate）；

3）调用 commlteMIMO _ updata 函数，根据新的调制类型更新所有参数。

Algorithm

MATLAB script segment

```
%% One subframe step processing
[dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
   commlteMIMO_SM_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
 %% Report average data rates
ADR=zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
%% Calculate  bit errors
Measures = step(hPBer, dataIn, dataOut);
%% Visualize results
if (visualsOn && prmLTEPDSCH.Eqmode~=3)
   zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
end;
% Change of modulation and coding rates randomly
Average_cRate=0.4932;
modType=randi([1 3],1,1);
cRate = Average_cRate + (1/6)*randn;
if cRate > 0.95, cRate=0.95;end; if cRate < 1/3, cRate=1/3;end;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_update( ...
   prmLTEPDSCH, prmLTEDLSCH, prmMdl, modType, cRate);
% Update subframe number
nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
```

7.6.3 基于 CQI 的自适应

下面的 MATLAB 代码为 While 循环体内的部分。在进行与无自适应情况相同的处理之外，While 循环体进行如下三个操作：

1）调用 CQIselection 和 CQI2indexMCS 函数生成 CQI 报告，对后面的子帧估计最优信道方案（modType）和码率（cRate）；

2）根据新的调制类型，调用 commlteMIMO _ update 函数更新 LTEPDSCH，LTEDLSCH，以及 prmMdl 参数；

3）调用 zVisSinr 函数描 24 个 SINR 测量的过去值，可视化信道质量随时间

的变化关系。

Algorithm

MATLAB script segment

```
%% One subframe step processing
[dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr] = ...
    commlteMIMO_SM_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
%% Report average data rates
ADR=zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
%% CQI feedback
sinr=CQIselection(dataOut, yRec,  nS, prmLTEDLSCH, prmLTEPDSCH);
[modType, cRate]=CQI2indexMCS(sinr);
%% Calculate  bit errors
Measures = step(hPBer, dataIn, dataOut);
%% Visualize results
if (visualsOn && prmLTEPDSCH.Eqmode~=3)
    zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
    zVisSinr(sinr);
end;
% Adaptive change of modulation and coding rate
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_update(...
    prmLTEPDSCH, prmLTEDLSCH, prmMdl, modType, cRate);
% Update subframe number
nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
```

7.6.4 收发端性能验证

仿真所用的所有参数总结在下面的 commlteMIMO _ params 脚本中。在仿真过程中，除了调制类型 modType 之外，其他参数为常量。通过执行 MIMO 接收器自适应调制和编码率模型的测试脚本，我们可以观察评估系统性能。在仿真运行中 MATLAB 的信息如图 7.3 所示。我们可以看到，同时改变调制方案和码率会影响瞬态数据速率，并最终影响收发端的平均数据速率。

7.6.5 结论

针对每种自适应情况，我们分别处理 2000 万比特以计算 BER。对前四种仿真（无自适应情况下的 QPSK、16QAM，和 64QAM，以及随机选择调制情况），码率恒定 0.4932。该码率为自适应编码的平均码率。为了方便比较，我们选择其作为随机情况的码率。结果总结在表 7.3 中。

```
Modulation                        = 64QAM
Instantanous Data rate            = 61.66 Mbps
Average Data rate                 = 61.66 Mbps
Instantanous Modulation rate = 6.00
Average Modulation rate           = 6.00
Instantanous Coding rate          = 0.3333
Average Coding rate               = 0.3333

Modulation                        = 16QAM
Instantanous Data rate            = 39.23 Mbps
Average Data rate                 = 50.45 Mbps
Instantanous Modulation rate = 4.00
Average Modulation rate           = 5.00
Instantanous Coding rate          = 0.3685
Average Coding rate               = 0.3509

Modulation                        = QPSK
Instantanous Data rate            = 31.70 Mbps
Average Data rate                 = 44.20 Mbps
Instantanous Modulation rate = 2.00
Average Modulation rate           = 4.00
Instantanous Coding rate          = 0.6225
Average Coding rate               = 0.4415
```

图 7.3　自适应调制和编码：数据速率、调制和码率模式
（注：图中的“bps”即为“bit/s”）

表 7.3　自适应调制和编码：不同情况的 BER、数据速率、调制和码率

调制类型	平均数据速率/(Mbit/s)	调制率	码率	比特误码率
QPSK - 无自适应	28.34	2	0.4932	2.8e^{-06}
16QAM - 无自适应	57.34	4	0.4932	7.9e^{-04}
64QAM - 无自适应	87.01	6	0.4932	3.6e^{-02}
随机选择	56.81	2 或 4 或 6	0.5037	2.5e^{-02}
自适应调制和编码	64.73	2 或 4 或 6	0.333 ~ 0.94	4.7e^{-03}

自适应调制和编码的结果与只使用自适应调制的情况类似。固定调制和编码率，我们可得到高数据速率和高阶调制，但严重损失 BER 性能。通过随机改变调制并固定码率，我们得到一个略好的结果。而基于信道质量的自适应调制和编码则能得到最好的折中。CQI 自适应技术的数据速率（64.73Mbit/s）高于随机选择的情况（56.81Mbit/s）。自适应编码的 BER（0.0047）低于随机选择的情

况（0.0250）。

7.7 自适应预编码

在本节中，我们在连续的子帧间用 PMI 信道状态报告自适应改变预编码矩阵索引。该自适应技术只适用于闭环空分复用传输模式。我们用一个布尔值参数 enPMIfback，控制启用或关闭 PMI 机制。当这个值为真时，接收端开启 PMI 索引选择，并向发射端反馈以用于下一个子帧的处理。在仿真中我们使用固定的用户码书索引。反馈间隔为每子帧一次（宽带）。

Algorithm

MATLAB function

```
function [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr_ref, cbIdx]...
   = commlteMIMO_SM_PMI_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl)
%% TX
persistent hPBer1
if isempty(hPBer1), hPBer1=comm.ErrorRate; end;
% Generate payload
dataIn = genPayload(nS,  prmLTEDLSCH.TBLenVec);
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
[data, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH, prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
% Map modulated symbols  to layers
numTx=prmLTEPDSCH.numTx;
LayerMapOut = LayerMapper(modOut, [], prmLTEPDSCH);
usedCbIdx = prmMdl.cbIdx;
% Precoding
[PrecodeOut, Wn] = SpatialMuxPrecoder(LayerMapOut, prmLTEPDSCH, usedCbIdx);
% Generate Cell-Specific Reference (CSR) signals
csr = CSRgenerator(nS, numTx);
csr_ref=complex(zeros(2*prmLTEPDSCH.Nrb, 4, numTx));
for m=1:numTx
   csr_pre=csr(1:2*prmLTEPDSCH.Nrb,:,:,m);
   csr_ref(:,:,m)=reshape(csr_pre,2*prmLTEPDSCH.Nrb,4);
```

```
end
% Resource grid filling
txGrid = REmapper_mTx(PrecodeOut, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
%% Channel
% MIMO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
% Add AWG noise
sigPow = 10*log10(var(rxFade));
nVar = 10.^(0.1.*(sigPow-snrdB));
rxSig =  AWGNChannel(rxFade, nVar);
%% RX
% OFDM Rx
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = REdemapper_mTx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
   chEst = ChanEstimate_mTx(prmLTEPDSCH, csrRx,  csr_ref, prmMdl.chEstOn);
   hD     = ExtChResponse(chEst, idx_data, prmLTEPDSCH);
else
   idealChEst = IdChEst(prmLTEPDSCH, prmMdl, chPathG);
   hD =  ExtChResponse(idealChEst, idx_data, prmLTEPDSCH);
end
% PMI codebook selection
if (prmMdl.enPMIfback)
cbIdx = PMICbSelect( hD, prmMdl.enPMIfback, prmLTEPDSCH.numTx, ...
   prmLTEPDSCH.numLayers, nVar );
else
   cbIdx=prmMdl.cbIdx;
end
% Frequency-domain equalizer
if (numTx==1)
   % Based on Maximum-Combining Ratio (MCR)
   yRec = Equalizer_simo(dataRx, hD, nVar, prmLTEPDSCH.Eqmode);
else
   % Based on Spatial Multiplexing
   yRec = MIMOReceiver(dataRx, hD, prmLTEPDSCH, nVar, Wn);
end
% Demap received codeword(s)
[cwOut, ~] = LayerDemapper(yRec, prmLTEPDSCH);
if prmLTEPDSCH.Eqmode < 3
   % Demodulate
   demodOut = DemodulatorSoft(cwOut, prmLTEPDSCH.modType, max(nVar));
else
```

```
    demodOut = cwOut;
end
% Descramble received codeword
rxCW =  lteDescramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes - CB segmentation, turbo decoding, rate dematching
[decTbData1, ~,~] = lteTbChannelDecoding(nS, rxCW, Kplus1, C1,  prmLTEDLSCH,
prmLTEPDSCH);
% Transport block CRC detection
[dataOut, ~] = CRCdetector(decTbData1);
end
```

7.7.1 基于 PMI 的自适应

下面的 MATLAB 代码为 While 循环体内的部分。代码实现 PMI 码书索引选择，但并不进行自适应调制和编码。commlteMIMO _ SM _ PMI _ step 函数负责在当前帧更新 PMI 码书索引，我们将这个函数的最后一个输出变量（cbIdx），作为 commlteMIMO _ update 函数的第三个输入，设定下一子帧的 PMI 索引。

Algorithm

MATLAB script segment

```
%% One subframe step processing
    [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr_ref, cbIdx]...
    = commlteMIMO_SM_PMI_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
     %% Report average data rates
    ADR=zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
    fprintf(1,'PMI codebook index = %2d\n', cbIdx);
    %% Calculate  bit errors
    Measures = step(hPBer, dataIn, dataOut);
    %% Visualize results
    if (visualsOn && prmLTEPDSCH.Eqmode~=3)
        zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
    end;
    %% Update subframe number
    nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
    % Adaptive PMI
    modType=prmLTEPDSCH.modType;
    cRate=prmLTEDLSCH.cRate;
    [prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_update(...
        prmLTEPDSCH, prmLTEDLSCH, prmMdl, modType, cRate, cbIdx);
    %% including adaptations here
```

7.7.2 收发端性能验证

仿真所用的所有参数总结在下面的 commlteMIMO _ params 脚本中。在仿真过程中，除了 PMI 码书索引（cbIdx）之外，其他参数为常量。通过执行 MIMO 接收器模型测试脚本，我们可以观察评估系统性能。在仿真运行中 MATLAB 的信息如图 7.4 所示，我们可以看到 PMI 码书索引的变化。

```
PMI codebook index =  1
Modulation                        = 16QAM
Instantanous Data rate            = 19.85 Mbps
Average Data rate                 = 19.85 Mbps
Instantanous Modulation rate = 4.00
Average Modulation rate           = 4.00
Instantanous Coding rate          = 0.3333
Average Coding rate               = 0.3333

PMI codebook index =  2
Modulation                        = 16QAM
Instantanous Data rate            = 19.85 Mbps
Average Data rate                 = 19.85 Mbps
Instantanous Modulation rate = 4.00
Average Modulation rate           = 4.00
Instantanous Coding rate          = 0.3333
Average Coding rate               = 0.3333
```

图 7.4 自适应 PMI：改变 PMI 码书索引

（注：图中的“bps”即为 bit/s）

表 7.4 自适应预编码：不同情况的 BER、数据速率、调制和码率

调制类型	平均数据速率/(Mbit/s)	调制率	码率	比特误码率
无自适应调制和编码	35.16	4	1/3	0.1278
无自适应调制和编码 + 自适应 PMI	35.16	4	1/3	0.01191

7.7.3 结论

针对每种自适应情况，我们分别处理 2000 万比特以计算 BER。结果在表 7.4 中。PMI 索引的变化不会影响收发端的调制方案、码率，瞬态数据速率或平均数据速率。而 BER 性能上如我们所预期一样，体现了自适应预编码的优势。

7.8 自适应 MIMO

在本节中，我们用 RI 信道状态报告自适应切换发射分集和空分复用传输模式。假如秩估计等于发射天线数，我们使用空分复用。当秩小于发射天线数时，我们选择发射分集模式。为了简单起见，我们使用相同的天线数，但我们不会用空分复用增加数据速率，以增大发射分集的链路可靠性。

通过引入状态数阈值，我们对所有子帧使用一个宽带秩估计值。函数输入为接收端 2D 信道矩阵（h）。这个矩阵根据传输使用双天线或四天线，配置 2 ×2 或 4 ×4 矩阵。

Algorithm

MATLAB function

```
function [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr_ref, cbIdx, ri]...
    = commlteMIMO_SM_PMI_RI_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH,
prmMdl)
%% TX
persistent hPBer1
if isempty(hPBer1), hPBer1=comm.ErrorRate; end;
%  Generate payload
dataIn = genPayload(nS,  prmLTEDLSCH.TBLenVec);
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
[data, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH, prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
% Map modulated symbols  to layers
if (prmLTEPDSCH.txMode ==4)
LayerMapOut = LayerMapper(modOut, [], prmLTEPDSCH);
usedCbIdx = prmMdl.cbIdx;
% Precoding
```

```
[PrecodeOut, Wn] = SpatialMuxPrecoder(LayerMapOut, prmLTEPDSCH, usedCbIdx);
else
% TD with SFBC
PrecodeOut = TDEncode(modOut(:,1),prmLTEPDSCH.numTx);
end
% Generate Cell-Specific Reference (CSR) signals
numTx=prmLTEPDSCH.numTx;
csr = CSRgenerator(nS, numTx);
csr_ref=complex(zeros(2*prmLTEPDSCH.Nrb, 4, numTx));
for m=1:numTx
   csr_pre=csr(1:2*prmLTEPDSCH.Nrb,:,:,m);
   csr_ref(:,:,m)=reshape(csr_pre,2*prmLTEPDSCH.Nrb,4);
end
% Resource grid filling
txGrid = REmapper_mTx(PrecodeOut, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
%% Channel
% MIMO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
% Add AWG noise
sigPow = 10*log10(var(rxFade));
nVar = 10.^(0.1.*(sigPow-snrdB));
rxSig =  AWGNChannel(rxFade, nVar);
%% RX
% OFDM Rx
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = REdemapper_mTx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
   chEst = ChanEstimate_mTx(prmLTEPDSCH, csrRx,  csr_ref, prmMdl.chEstOn);
   hD     = ExtChResponse(chEst, idx_data, prmLTEPDSCH);
else
   idealChEst = IdChEst(prmLTEPDSCH, prmMdl, chPathG);
   hD =  ExtChResponse(idealChEst, idx_data, prmLTEPDSCH);
end
% Frequency-domain equalizer
if (numTx==1)
   % Based on Maximum-Combining Ratio (MCR)
   yRec = Equalizer_simo(dataRx, hD, nVar, prmLTEPDSCH.Eqmode);
else
   if (prmLTEPDSCH.txMode ==4)
      % Based on Spatial Multiplexing
      [yRec, ri] = MIMOReceiver_ri(dataRx, hD, prmLTEPDSCH, nVar, Wn);
   else
% Based on Transmit Diversity  with SFBC combiner
  [yRec, ri] = TDCombine_ri(dataRx, hD, prmLTEPDSCH.numTx, prmLTEPDSCH.numRx);
   end
```

```
end
% PMI codebook selection
if (prmMdl.enPMIfback)
cbIdx = PMICbSelect( hD, prmMdl.enPMIfback, prmLTEPDSCH.numTx, ...
   prmLTEPDSCH.numLayers, snrdB);
else
   cbIdx = prmMdl.cbIdx;
end
% Demap received codeword(s)
[cwOut, ~] = LayerDemapper(yRec, prmLTEPDSCH);
if prmLTEPDSCH.Eqmode < 3
   % Demodulate
   demodOut = DemodulatorSoft(cwOut, prmLTEPDSCH.modType, max(nVar));
else
   demodOut = cwOut;
end
% Descramble received codeword
rxCW =  lteDescramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes - CB segmentation, turbo decoding, rate dematching
[decTbData1, ~,~] = lteTbChannelDecoding(nS, rxCW, Kplus1, C1,  prmLTEDLSCH,
prmLTEPDSCH);
% Transport block CRC detection
[dataOut, ~] = CRCdetector(decTbData1);
end
```

7.8.1 基于 RI 的自适应

下面的 MATLAB 代码为 While 循环体内的部分。代码实现基于 RI 的自适应，但并不进行前面所讲到的其他自适应技术。在当前子帧，commlteMIMO _ SM _ RI _ step 函数的最后一个输出变量（ri）负责更新秩估计索引。我们将这个索引值作为 commlteMIMO _ update 函数的第四个输入，设定下一子帧的 RI 索引。

Algorithm

MATLAB script segment

```
%% One subframe step
   [dataIn, dataOut, txSig, rxSig, dataRx, yRec, csr, cbIdx, ri] ...
   = commlteMIMO_SM_PMI_RI_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH,
prmMdl);

   Ri=RIselection(ri, threshold);
```

```
ADR_a=zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
 fprintf(1,'PMI codebook index = %2d\nTransmission mode  = %2d\n', cbIdx, Ri);

%% Calculate  bit errors
Measures = step(hPBer, dataIn, dataOut);
%% Visualize results
if (visualsOn && prmLTEPDSCH.Eqmode~=3)
    zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
end;
% Update subframe number
nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
% Adaptive RI
modType=prmLTEPDSCH.modType;
cRate=prmLTEDLSCH.cRate;
cbIdx=prmMdl.cbIdx;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_update(...
   prmLTEPDSCH, prmLTEDLSCH, prmMdl, modType, cRate,  cbIdx, Ri);
```

7.8.2 收发端性能验证

仿真所用的所有参数总结在下面的 commlteMIMO _ params 脚本中。在仿真过程中，除了秩估计索引（ri）之外，其他参数为常量。在仿真运行中 MATLAB 的信息如图 7.5 所示，我们可以看到传输模式根据 RI 索引改变，影响瞬态数据速率，并最终影响收发端平均数据速率。

```
PMI codebook index =  1
Transmission mode  =  4
Modulation                       = 16QAM
Instantanous Data rate           = 19.85 Mbps
Average Data rate                = 19.14 Mbps
Instantanous Modulation rate = 4.00
Average Modulation rate          = 4.00
Instantanous Coding rate         = 0.3333
Average Coding rate              = 0.3333

PMI codebook index =  1
Transmission mode  =  2
Modulation                       = 16QAM
Instantanous Data rate           = 9.91 Mbps
Average Data rate                = 19.03 Mbps
Instantanous Modulation rate = 4.00
Average Modulation rate          = 4.00
Instantanous Coding rate         = 0.3333
Average Coding rate              = 0.3333
```

图 7.5 自适应性 MIMO：改变传输模式

（注：图中的 “bps” 即为 “bit/s”）

7.8.3 结论

如我们预期，我们使用空分复用可以得到比发射分集更高的数据速率（见表7.5）。通过秩估计方法，我们得到RI自适应技术的平均数据速率与空分复用接近。这个结果也反映了满秩占的比例（86.5%）比低秩（13.5%）更多。

表7.5 自适应性秩估计和MIMO：不同情况的BER、数据速率、调制和编码率

调制类型	平均数据速率/(Mbit/s)	调制率	码率	比特误码率
固定模式：发射分集	15.26	4	2/3	3.4e-07
固定模式：空分复用	19.85	4	1/3	1.3e-03
RI反馈自适应模式	19.23	4	1/3	7.1e-04

7.9 下行链路控制信息

如前文所述，链路自适应包含下面三个部分：

1）接收端信道状态估计；

2）调度操作生成调度分配；

3）在下一个子帧传输调度分配。

在下行链路中，调度分配包含在DCI中。LTE传输模式定义了多个DCI格式。一种DCI格式可能含有各种信息，包括多天线属性如预编码矩阵指示索引、调制方案，和传输块大小；以及HARQ（混合式自动重传请求）处理内容如处理数、冗余版本，和最新数据指示。

在本书中，我们关注用户层信息和单用户处理。因此，我们不打算详细讨论DCI格式及其MATLAB实现。不过我们会讲解两个与控制信息有关的要点：

1）CQI信息如何映射到MCS信息；

2）如何在发送之前将PDCCH编码放入下行链路资源网格的前几个OFDM符号。最后我们通过和前面讲到的PDSCH比较BER性能，快速回顾PDSCH处理的可靠性。

7.9.1 MCS

在下行链路发射端，调度器为当前子帧分配调制类型和码率。调度信息以5bit的MCS索引编码并封装入不同的DCI格式中。MCS索引连带编码调制方案和传输块大小（tbSize）。码率（R）定义为输入比特数（K）和输出比特数（N）之比：K/N。因DLSCH处理之前要添加24比特CRC，故输入比特数为tb-

Size + 24。DLSCH 编码器输出比特数 N 等于每个 PDSCH 码字的长度（numPDSCHbits），即 $N = \text{numPDSCHbits}$。又因为 PDSCH 码字长度由资源区块数完全确定，故编码率可由传输块大小（tbSize）确定：$R = \dfrac{\text{tbSise} + 24}{\text{numPDSCHbits}}$

下面的 MATLAB 函数说明了 CQI 信息（调制方案和目标码率）如何映射传输块大小、传输块查询表索引，和实际码率。

Algorithm

MATLAB function

```
function [tbSize, TBSindex, ActualRate] = getTBsizeMCS(modType, TCR, Nrb,
numLayers, numPDSCHBits)
% Get the transport block size for a specified configuration.
% Inputs:
%   modType     : 1 (QPSK), 2 (16QAM), 3 (64QAM)
%   TCR         : Target Code Rate
%   Nrb         : number of resource blocks
%   numLayers   : number of layers
%   numPDSCHBits: number of PDSCH bits (G)
% Output:
%   tbSize : transport block length
% Example: R.10 of A.3.3.2.1 in 36.101
%   tbLen =  getTBsizeRMC(1, 1/3, 50, 1, 12384)
% Reference:
%   1) Section 7.1.7 of 36.213, for TB sizes.
%      Uses preloaded Tables 7.1.7.1-1, 7.1.7.2.1-1, 7.1.7.2.2 and 7.1.7.2.5.
%   2) Section A.3.1 of 36.101 for TB size selection criteria.
switch modType
   case 1 % QPSK
      numTBSizes = 10;
      stIdx = 0;
   case 2 % 16QAM
      numTBSizes = 7;
      stIdx = 9;
   case 3 % 64QAM
      numTBSizes = 12;
      stIdx = 15;
end
numBitsPerLayer=numPDSCHBits/numLayers;
% Load saved entries for Tables 7.1.7.2.1-1, 7.1.7.2.2 and 7.1.7.2.5.
load TBSTable.mat
tbVec = baseTBSTab(stIdx+(1:numTBSizes), Nrb);      % for 1-based indexing
ProposedRates=(tbVec+24)./numBitsPerLayer;
ProposedRates(ProposedRates<1/3)=10;
```

```
[~, c] = min(abs(TCR-ProposedRates));
tbSize = tbVec(c);
if (numLayers==2) % Section 7.1.7.2.2
   if (Nrb <= 55)
      tbSize = baseTBSTab(TBSindex, 2*Nrb);
   else
      index=(layer2TBSTab(:,1)==tbSize);
      tbSize = layer2TBSTab(index, 2);
   end
elseif (numLayers==4) % Section 7.1.7.2.5
   if (Nrb <= 27)
      tbSize = baseTBSTab(TBSindex, 4*Nrb);
   else
      index=(layer4TBSTab(:,1)==tbSize);
      tbSize = layer4TBSTab(index, 2);
   end
end
ActualRate=(tbSize+24)./numPDSCHBits;
TBSindex= stIdx+c;
```

下面的函数说明了传输块大小索引和调制类型如何映射一个 5bit 标量量化器编码的 MCS 索引。

Algorithm

MATLAB function

```
function MCSindex=map2MCSindex(TBSindex, modType)
%#codegen
% Assume 1-based indexing
if ((TBSindex < 1) || (TBSindex >27)),
error('map2MCSindex function: Wrong TBSindex.');end
switch TBSindex
   case 10
      switch modType
         case 1
            MCSindex=10;
         case 2
            MCSindex=11;
         otherwise
            error('Wrong combination of TBSindex and modulation type');
      end
   case 16
      switch modType
         case 2
            MCSindex=17;
         case 3
```

```
            MCSindex=18;
         otherwise
            error('Wrong combination of TBSindex and modulation type');
      end
   otherwise
      if TBSindex <10
         MCSindex=TBSindex;
      elseif ((TBSindex >10) && (TBSindex <16))
         MCSindex=TBSindex+1;
      else
         MCSindex=TBSindex+2;
      end
end
```

7.9.2 自适应率

调度决策可通过更新每一个子帧影响下行链路。但比如像包含 MCS 的 DCI 则并不需要每 1ms 都进行自适应。调度算法并不包括控制码率自适应间隔。这个指标受多个参数控制，包括全部用户的链路质量、基站交调、服务质量要求、服务，和服务优先级[9]。

7.9.3 DCI 处理

在本节中我们将考察使用收发端应用 DCI 处理的性能。DCI 编码放入每个子帧的前几个 OFDM 符号并发送。DCI 的可靠译码是从子帧 OFDM 符号上正确复原用户数据的关键。因 DCI 通常是小包，故一般使用卷积码对其编码。LTE 标准定义一种尾比特卷积编码，并以发射分集保证 DCI 传递的链路质量信息。

因我们只关注用户层处理，故我们不会讨论 DCI 包的详细内容（包括物理混合 ARQ 指示信道（PHICH）和物理控制格式指示信道（PCFICH））。我们也不会关注 DCI 信息在资源网格和 OFDM 传输中的位置。我们会关注信号通过平坦信道和 AWGN（加性白高斯噪声）信道进行传输之前的信号处理链，以评估控制信息的性能。

7.9.3.1 收发器函数

下面的 MATLAB 函数（commlteMIMO _ DCI _ step）总结了 DCI 处理链。在发射端，我们首先生成 DCI 特定的 CRC。DCI 数据通过尾比特卷积编码器进行编码。这个编码器和用户层 Turbo 编码器有相同的网状结构。随后我们将编码比特进行码率匹配、绕码，和调制，这一过程与用户数据比特处理相同。最后，用发射分集处理调制信号。在这个函数中我们不考虑资源网格映射和 OFDM 信号生成。我们将发射分集编码器的输出直接输入信道模型。在接收端，我们反向发射端的操作，包括理想信道估计、发射分集合并、软判决译码、解绕码、码率解匹

配，和 Viterbi 译码。最后，通过 DCI 特定 CRC 校验，我们可以得到 DCI 比特的最优估计输出。函数 commlteMIMO _ DCI _ step 概括了一个收发端应用 DCI 处理的最简单模型。

Algorithm

MATLAB function

```
function [dataIn, dataOut, modOut, rxSig]...
   = commlteMIMO_DCI_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl)
%% TX
numBitsDCI = 205;
%  Generate payload
dataIn        = genPayload(nS,  numBitsDCI);
% Transport block CRC generation
CrcOut1       = CRCgeneratorDCI(dataIn);
% Channel coding includes - tail biting convolutional coding, rate matching
data           = TailbitingConvEnc(CrcOut1,  prmLTEDLSCH.cRate);
%Scramble codeword
scramOut     = lteScramble(data, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut       = Modulator(scramOut, prmLTEPDSCH.modType);
% Transmit diversity encoder
PrecodeOut = TDEncode(modOut, prmLTEPDSCH.numTx);
%% Channel
% MIMO fading channel
[fadeOut, pathGain]    = MIMOFadingChan(PrecodeOut, prmLTEPDSCH, prmMdl);
nVar                         = real(var(fadeOut(:)))/(10.^(0.1*snrdB));
pathG                             = squeeze(pathGain);
% AWGN
recOut                            = AWGNChannel(fadeOut, nVar);
%% RX
% Transmit diversity combiner
rxSig =  TDCombine(recOut, pathG,  prmLTEPDSCH.numTx,  prmLTEPDSCH.numRx);
% Demodulate
demodOut = DemodulatorSoft(rxSig, prmLTEPDSCH.modType, nVar);
% Descramble both received codewords
rxCW1 =  lteDescramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
% Channel decoding includes - tail biting Viterbi decoding, rate dematching
L=numel(CrcOut1);
decData1=TailbitingViterbiSoft( rxCW1, L);
% Transport block CRC detection
[dataOut, ~] = CRCdetectorDCI(decData1);
end
```

收发器函数调用两个函数：TailbitingconvEnc 和 tailbitingViterbiSoft。这两个函数用通信系统工具箱的系统对象实现尾比特卷积编码和反 Viterbi 译码。在编码器中，我们创建两个卷积编码器：一个用来保持编码器状态，而另一个依据编

码器状态对数据帧进行流处理。最后，我们进行码率匹配操作得到输出。

Algorithm

MATLAB function

```
function y=TailbitingConvEnc(u, codeRate)
%#codegen
trellis=poly2trellis(7, [133 171 165]);
L=numel(u);
C=6;
persistent ConvEncoder1 ConvEncoder2
if isempty(ConvEncoder1)
   ConvEncoder1=comm.ConvolutionalEncoder('TrellisStructure', trellis,
'FinalStateOutputPort', true, ...
      'TerminationMethod','Truncated');
   ConvEncoder2 =  comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'InitialStateInputPort', true,...
      'TrellisStructure', trellis);
end
u2            = u((end-C+1):end);                    % Tail-biting convolutional coding
[~, state]  = step(ConvEncoder1, u2);
u3            = step(ConvEncoder2, u,state);
y             = fcn_RateMatcher(u3, L, codeRate);  % Rate matching
```

在译码器中，我们首先进行码率解匹配，随后进行接收信号似然估计并进行软判决 Viterbi 译码。最后通过抽取 Viterbi 译码输出样本得到译码器输出。

Algorithm

MATLAB function

```
function y=TailbitingViterbiSoft(u, L)
%#codegen
trellis=poly2trellis(7, [133 171 165]);
Index=[L+1:(3*L/2) (L/2+1):L];
persistent Viterbi
if isempty(Viterbi)
   Viterbi=comm.ViterbiDecoder(...
      'TrellisStructure', trellis,
'InputFormat','Unquantized','TerminationMethod','Truncated','OutputDataType','logical');
end
uD           = fcn_RateDematcher(u, L);                 % Rate de-matching
uE           = [uD;uD];                                  % Tail-biting
uF           = step(Viterbi, uE);                        % Viterbi decoding
y            = uF(Index);
```

7.9.3.2 DCI 收发器测试脚本

下面的测试脚本函数（commlteMIMO _ DCI）评估 DCI 收发端性能。函数输

入为 Eb/N0 值、规定的最大误码数，和最大处理比特数。程序输出为 BER 测量和实际处理比特数。首先，程序调用初始化函数（commlteMIMO _ initialize）设定所有相关参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。然后程序用一个 While 循环调用 DCI 处理链函数处理子帧。

Algorithm

MATLAB function

```
function [ber, bits]=commlteMIMO_DCI(EbNo, maxNumErrs, maxNumBits)
%
clear functions
%% Set simulation parameters & initialize parameter structures
commlteMIMO_ params_DCI;
codeRate=cRate;
k=2*modType;
snrdB = EbNo + 10*log10(codeRate) + 10*log10(k);
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteMIMO_initialize(txMode, ...
   chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode, chan-
Mdl, Doppler, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx, snrdB, maxNumErrs, maxNumBits);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl Doppler corrLvl chEstOn numCodeWords enPMIfback cbIdx
%%
hPBer = comm.ErrorRate;
%% Simulation loop
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
  [dataIn, dataOut, modOut, rxSig] = ...
     commlteMIMO_DCI_step(nS, snrdB, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
   % Calculate  bit errors
   Measures = step(hPBer, dataIn, dataOut);
   % Visualize results
   if visualsOn, zVisConstell( prmLTEPDSCH, modOut, rxSig, nS); end;
end
ber=Measures(1);
bits=Measures(3);
reset(hPBer);
```

7.9.3.3 BER 测量

DCI 收发器的特点为应用低星座点密度调制（QPSK）、尾比特卷积码编码和发射分集编码。如图 7.6 所示，这个 DCI 收发器处理 DCI 信息有较高的 BER 性能，如 LTE 标准设计要求。高 BER 性能这个指标非常关键，因为控制信息错误过多会为复原用户数据带来灾难性后果。

图 7.6 表示了 Eb/N0 值在 0 ~ 4dB 之间的 BER 结果。程序设定最大误码数

为 1000，最大处理比特数为 $1e^7$。我们在 CSI 完全可知的情况下，通过平坦衰落信道传输数据。这标志着可以进行理想信道估计。注意 BER 测量值即使在低 SNR（信号噪声比）值时也很低。比如，对 Eb/N0 为 0dB 的情况，BER 为 $1e^{-4}$，而 2dB 的 Eb/N0 时，BER 为 $1e^{-6}$。

第 4 章讨论的 PDSCH 处理性能可以与这个结果可以媲美。即使 PDSCH 处理不使用发射分集，仅仅通过 LTE Turbo 编码、绕码，和调制等处理，我们也可以得到与 DCI 处理相比肩的 BER 性能。

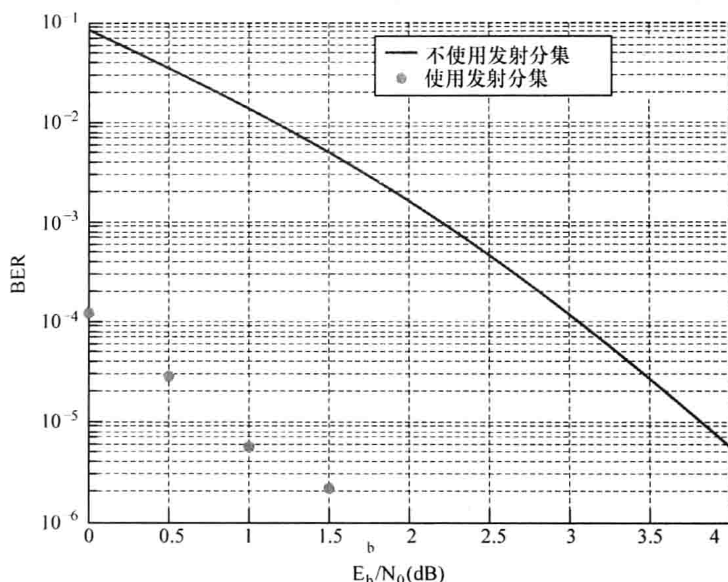


图 7.6 DCI 的 BER 性能：通过 AWGN 和平坦衰落信道传输并进行理想信道估计

7.10 本章小结

在本章中，我们研究了一些 LTE 标准中定义的链路自适应技术。这些链路自适应技术工作在接收端并包括选择和反馈多种系统参数。发射端处理下一子帧时应用这些参数。这些系统参数包括：

1）调制和编码方案；

2）空分复用的传输层数；

3）闭环空分复用模式中预编码矩阵的选择。链路自适应通常以提高用户数据速率和增大频谱效率为参数选择标准。

我们回顾了自适应所需要的信道质量测量。它们包括 CQI、PMI，和 RI 测量。我们随后介绍了基于信道质量测量实现自适应的算法，包括自适应调制和编

码、自适应预编码确定最优预编码矩阵，和解决空分复用模式秩缺陷问题的自适应 MIMO。最后，我们回顾了包括 DCI 传输的信号处理链。我们看到通过采用发射分集和尾比特卷积编码，LTE 标准提供了一种控制信息传输的可靠机制。

参考文献

[1] 3GPP (2009) Requirements for Evolved UTRA (E-UTRA) and Evolved UTRAN (E-UTRAN) v9.0.0. TR 25.913, December 2009.

[2] Ding, L., Tong, F., Chen, Z., and Liu, Z. (2011) A novel MCS selection criterion for VOIP in LTE. International Conference on Wireless Communications, Networking and Mobile Computing - WiCom, pp. 1–4.

[3] Pande, A., Ramamurthi, V., and Mohapatra, P. (2011) Quality-oriented video delivery over LTE using adaptive modulation and coding. IEEE Global Telecommunications Conference (GLOBECOM), pp. 1–5.

[4] Tan, P., Wu, Y. and Sun, S. (2008) Link adaptation based on adaptive modulation and coding for multiple-antenna OFDM system. *IEEE Journal on Selected Areas in Communications*, **26**, 8, 1599–1606.

[5] Kim, J., Lee, K., Sung, C. and Lee, I. (2009) A simple SNR representation method for AMC schemes of MIMO systems with ML detector. *IEEE Transactions on Communications*, **57**, 2971–2976.

[6] Flahati, S., Svensson, A., Ekman, T. and Sternad, M. (2004) Adaptive modulation systems for predicted wireless channels. *IEEE Transactions on Communications*, **52**, 307–316.

[7] Ohlmer, E. and Fettweis, G. (2009) Link adaptation in linearly precoded closed-loop MIMO-OFDM systems with linear receivers. IEEE International Conference on Communications (ICC), June 2009.

[8] Love, D. and Heath, R. (2005) Limited feedback unitary precoding for spatial multiplexing systems. *IEEE Transactions on Information Theory*, **51**, 8, 2967–2976.

[9] Ghosh, A. and Ratasuk, R. (2011) *Essentials of LTE and LTE-A*, Cambridge University Press.

[10] 3GPP (2013) Physical Layer Procedures v11.3.0. TR 25.213, June 2013.

[11] Jiang, M., Prasad, N., Yue, G., and Rangarajan, S. (2011) Efficient link adaptation for precoded multi-rank transmission and turbo SIC receivers. IEEE ICC, 2011.

8　系统级建模

到现在为止，我们分别研究了 LTE 标准的四个核心技术：OFDM（正交频分复用）多载波传输、MIMO（多输入多输出）多天线技术，Turbo 码信道编码，和链路自适应。下行链路的 OFDM 和上行链路的单载波 SC－FDM（单载波频分复用）构成了 LTE 标准传输策略的骨架。如第 7 章中讲到的，自适应调制和编码提供了高频谱效率。可以说 LTE 与过去的童鞋标准相比最大的进步是引入各种 MIMO 技术。对任意给定时间，LTE 下行链路收发器系统工作模式都可以归类到 LTE 的 9 种 MIMO 多天线技术中。因此，当我们评估 LTE 系统的性能时，我们应该更多关注所用的 MIMO 技术和其相应的信道工作条件。

在本章中，我们将所有 LTE 标准的物理层仿真模型统合在一起。在前面各章中，我们关注的单点传输模式并逐步构建模型。在本章中，我们将构建收发端多点传输模式的仿真模型。我们也将评估与系统性能相关的各个指标。

我们将会重点关注多点模式的一般处理和用于任意特定传输模式的特性。我们将继续关注用户层处理和单载波传输，仿真模型将包括 LTE 标准的前四种传输模式。随后我们将改变系统工作状态评估性能：包括考察使用不同 MIMO 模式时的系统质量和吞吐量，信道模型、信道估计技术，和 MIMO 接收器算法。最后，我们将建立 LTE PHY 系统 Simulink 模型。我们会展示 Simulink 模型如何和 MATLAB 算法协作，简单的创建系统模型。这也是本书一个新颖之处的所在。Simulink 模型的另一大优势在于其自身即为测试平台，而不需要单独开发 MATLAB 脚本在循环体内调用各个处理算法和可视化函数。我们将用 Simulink 完成所有任务。在本章最后，我们会对 LTE 模型进行一个定量评估。我们会用 LTE Simulink 模型处理声音流信号，将其输入下行链路发射端，并在一定处理延迟后，在接收端听到它并评估声音质量评估。

8.1　系统模型

在本节中我们整合各个核心技术，构建 LTE PHY 的系统模型。这个系统模型包括发射端、信道模型，和接收端。发射端处理按 LTE 标准定义。标准同时为性能评估定义了各种信道模型。为了让各个系统设计者有机会设计各不相同的方案实现不一样的性能指标，标准没有定义接收端处理。

图 8.1 所示为 LTE 下行链路收发端的总体结构。发射端负责处理来自传输

信道的载荷比特。发射端输出为一系列符号，它们由可用的发射天线发送。多发射天线发送的符号通过信道被接收天线接收，在每个接收天线生成接收符号集。接收端负责处理这些接收符号。通过一系列反向发射端的操作，接收端生成发射载荷比特的最优估计。

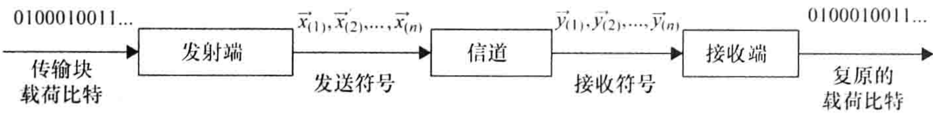


图 8.1　LTE 下行链路 PHY 收发端模型

8.1.1　发射端模型

发射端负责处理来自传输信道的载荷比特。信号处理根据传输模式的不同而不同。而传输模式则由下行链路调度器决定。传输模式决定选择处理给定子帧的 MIMO 技术。在本书中，我们关注 LTE 标准中九种传输模式的前四种。图 8.2 所示为下行链路发射端处理链。

传输块载荷比特　0100010011...

下行链路调度器

	模式 1 SIMO	模式 2 发射分集	模式 3 开环 空分复用	模式 4 闭环 空分复用
DLSCH 处理	CRC生成 Turbo编码 码率匹配	CRC生成 Turbo编码 码率匹配	CRC生成 Turbo编码 码率匹配	CRC生成(1,2) Turbo编码(1,2) 码率匹配(1,2)
PDSCH 处理	绕码 调制 小区特定参考生成 资源元素映射 OFDM传输	绕码 调制 发射分集编码 小区特定参考生成 资源元素映射 OFDM传输	绕码 调制 层映射 开环预编码 小区特定参考生成 资源元素映射 OFDM传输	绕码 (1,2) 调制 (1,2) 层映射 闭环预编码 小区特定参考生成 资源元素映射 OFDM传输

切换

发送符号　$\vec{x}_{(1)},\vec{x}_{(2)},\vec{x}_{(3)},\dots,\vec{x}_{(n)},\dots$

图 8.2　LTE 下行链路系统模型：传输模式 1～4，发射端操作

处理一个子帧时，调度器选择四种传输模式中的一种：第一种为单天线传输

的 SIMO 模式；第二种为通过多天线发送冗余信息提升链路可靠性的发射分集模式；第三种和第四种为提升数据速率的空分复用 MIMO。其中第三种模式为高移动率情况适用的开环空分复用。第四种为低移动率情况适用的闭环空分复用，这种模式可以实现 LTE 标准中的最高数据速率。

在每个传输模式中，我们通过下行链路公共信道（DLSCH）和下行链路公共物理信道（PDSCH）处理进行一系列操作。许多 PDSCH 和 PLSCH 操作对所有传输模式通用。各个传输模式中特殊的 MIMO 特征操作则各不相同。

通用操作包括传输块 CRC 添加、码块分割、Turbo 编码、码率匹配，和码率连接生成码字。LTE 下行链路定义支持一个或两个码字。为了使 MATLAB 算法简单易读，我们只研究闭环空分复用单码字或双码字的处理。在 PDSCH 中，通用操作为绕码、调制绕码后比特生成调制符号、映射调制符号到资源元素，以及生成 OFDM 信号以供每个天线端口发送。

不同的传输模式对应了相应的 MIMO 操作。在 SIMO 模式情况下，并没有特殊的 MIMO 操作且调制符号直接映射到资源网格。在第二种模式下，调制符号使用发射分集技术。该操作可以理解为层映射和预编码的组合。发射分集编码器将调制流按不同的发射天线分为不同的子流。这一过程与层映射类似。发射分集同时保证正交性的分配每个发射天线传输子流。这一过程可以认为是一种特殊的预编码。

在空分复用模式 3 和模式 4，我们对调制符号进行分立的层映射和预编码操作。随后，将 MIMO 操作的输出作为资源元素分配在资源网格中。在模式 3 中，发射端和传输端独立生成开环预编码使用的预编码矩阵，而不需要传输任何预编码数据。在闭环空分复用中，我们在仿真中使用恒定的预编码矩阵，或在接收端进行闭环反馈通信，确定哪个预编码矩阵索引需要在下一个子帧发送。

8.1.2 发射端模型的 MATLAB 模型

下面的 MATLAB 函数实现了任意给定子帧的传输模式下 LTE 下行链路发射操作。这部分操作组合了前面几张中详述的 SIMO、发射分集，开环和闭环空分复用。函数输入为子帧数（nS）和三个参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。每个帧操作都将调用函数，首先生成传输块载荷比特然后进行通用 DLSCH 和 PDSCH 处理。通过使用 MATLAB switch - case 声明，函数根据所选择的传输模式进行相应的 MIMO 操作。MIMO 输出符号和小区特征资源元素（导频）随后映射到资源网格。随后对资源网格进行 OFDM 发射操作最终生成输出发射符号（txSig）。

Algorithm

MATLAB function

```
function [txSig, csr_ref] = commlteMIMO_Tx(nS, dataIn, prmLTEDLSCH, prmLTEPDSCH,
prmMdl)
%#codegen
% Generate payload
dataIn1 = genPayload(nS, prmLTEDLSCH.TBLenVec);
% Transport block CRC generation
tbCrcOut1 =CRCgenerator(dataIn1);
% Channel coding includes - CB segmentation, turbo coding, rate matching,
% bit selection, CB concatenation - per codeword
[data1, Kplus1, C1] = lteTbChannelCoding(tbCrcOut1, nS, prmLTEDLSCH,
prmLTEPDSCH);
%Scramble codeword
scramOut = lteScramble(data1, nS, 0, prmLTEPDSCH.maxG);
% Modulate
modOut = Modulator(scramOut, prmLTEPDSCH.modType);
%%%%%%%%%%%%%%%%%%%%%%%%%
% MIMO transmitter based on the mode
%%%%%%%%%%%%%%%%%%%%%%%%%
numTx=prmLTEPDSCH.numTx;
dataIn= dataIn1;
Kplus=Kplus1;
C=C1;
Wn=complex(ones(numTx,numTx));
switch prmLTEPDSCH.txMode
   case 1 % Mode 1: Single-antenna (SIMO mode)
      PrecodeOut =modOut;

   case 2 % Mode 2: Transmit diversity
      % TD with SFBC
      PrecodeOut = TDEncode(modOut(:,1),prmLTEPDSCH.numTx);

   case 3 % Mode 3: Open-loop Spatial multiplexing
      LayerMapOut = LayerMapper(modOut, [], prmLTEPDSCH);
      % Precoding
      PrecodeOut = SpatialMuxPrecoderOpenLoop(LayerMapOut, prmLTEPDSCH);

   case 4 % Mode 4: Closed-loop Spatial multiplexing
      if prmLTEPDSCH.numCodeWords==1
         % Layer mapping
         LayerMapOut = LayerMapper(modOut, [], prmLTEPDSCH);
      else
         dataIn2 = genPayload(nS, prmLTEDLSCH.TBLenVec);
         tbCrcOut2 =CRCgenerator(dataIn2);
```

```
        [data2, Kplus2, C2] = lteTbChannelCoding(tbCrcOut2, nS, prmLTEDLSCH,
prmLTEPDSCH);
        scramOut2 = lteScramble(data2, nS, 0, prmLTEPDSCH.maxG);
        modOut2 = Modulator(scramOut2, prmLTEPDSCH.modType);
        % Layer mapping
        LayerMapOut = LayerMapper(modOut, modOut2, prmLTEPDSCH);
        dataIn= [dataIn1;dataIn2];
        Kplus=[Kplus1;Kplus2];
        C=[C1; C2];
    end
    % Precoding
    usedCbIdx = prmMdl.cbIdx;
    [PrecodeOut, Wn] = SpatialMuxPrecoder(LayerMapOut, prmLTEPDSCH, usedCbIdx);
end
% Generate Cell-Specific Reference (CSR) signals
numTx=prmLTEPDSCH.numTx;
csr = CSRgenerator(nS, numTx);
csr_ref=complex(zeros(2*prmLTEPDSCH.Nrb, 4, numTx));
for m=1:numTx
  csr_pre=csr(1:2*prmLTEPDSCH.Nrb,:,:,m);
  csr_ref(:,:,m)=reshape(csr_pre,2*prmLTEPDSCH.Nrb,4);
end
% Resource grid filling
txGrid = REmapper_mTx(PrecodeOut, csr_ref, nS, prmLTEPDSCH);
% OFDM transmitter
txSig = OFDMTx(txGrid, prmLTEPDSCH);
```

8.1.3 信道模型

信道建模包含了 MIMO 衰落信道和 AWGN（加性白高斯噪声）信道。MIMO 信道定义多发射天线信号与多接收天线信号间的关系。MIMO 信道典型参数包括天线配置、多径延迟属性、最大多普勒频移，和发射端接收端天线的空间相关性水平。AWGN 信道通常用 SNR（信噪比）值或噪声方差定义。图 8.3 表示了一个衰落信道模型下的操作，构建了一个 4×4 MIMO 信道，连接 4 组发射天线和 4 组接收天线间发射信号的单一路径。随后在每个 MIMO 接收信号中添加非相关性白高斯噪声得到信道建模输出信号。

8.1.4 信道模型的 MALTAB 模型

下面的 MATLAB 函数为多径 MIMO 衰落信道附加 AWGN 信道的信道模型。首先，通过调用 MIMOFadingChan 函数，我们生成经过衰落的发射信号（rxFade）和相应的信道矩阵（chPathG）。MIMO 衰落信道以多发射天线线性组合计算衰落信号。因此，输出信号（rxFade）不会包含一个天线的平均功率（信号方

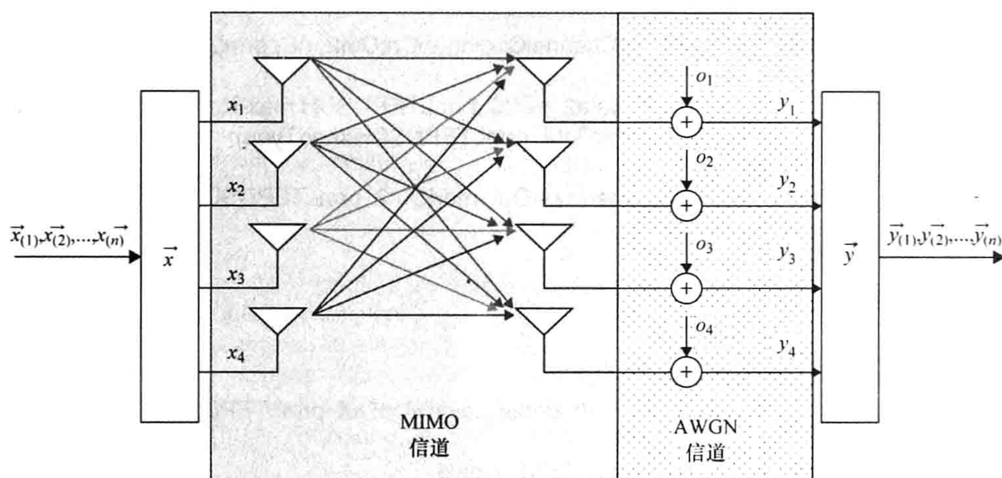


图 8.3　下行链路信道模型：单路径信道模型

差）。为了计算噪声发差，我们调用 AWGNChannel 函数，首先计算信号方差（sigPow），随后通过信号功率和 SNR 的 dB 值求差得到噪声方差。

Algorithm

MATLAB function

```
function [rxSig, chPathG, nVar] = commlteMIMO_Ch(txSig, prmLTEPDSCH, prmMdl )
%#codegen
snrdB = prmMdl.snrdB;
% MIMO Fading channel
[rxFade, chPathG] = MIMOFadingChan(txSig, prmLTEPDSCH, prmMdl);
% Add AWG noise
sigPow = 10*log10(var(rxFade));
nVar = 10.^(0.1.*(sigPow-snrdB));
rxSig =  AWGNChannel(rxFade, nVar);
```

8.1.5　接收端模型

信道模型后，信号处理链在接收端对接收符号进行处理。在接收端，基本上反向发射端操作以得到发射载荷比特的最优估计。图 8.4 所示为下行链路接收端处理链。

在接收端，前面的几个操作与传输模式无关。它们包括 OFDM 接收器、资源元素反映射，和小区特征参考（CSR）抽取。在这些通用操作之后，我们在接收端重建资源网格抽取用户数据和 CSR 信号。根据接收的 CSR 信号，我们随后估计每个子帧的信道响应矩阵。信道估计可以基于多个算法进行：在完全可知信道状态信息情况下进行理想估计，根据各种平均和内插形式进行其他方法的

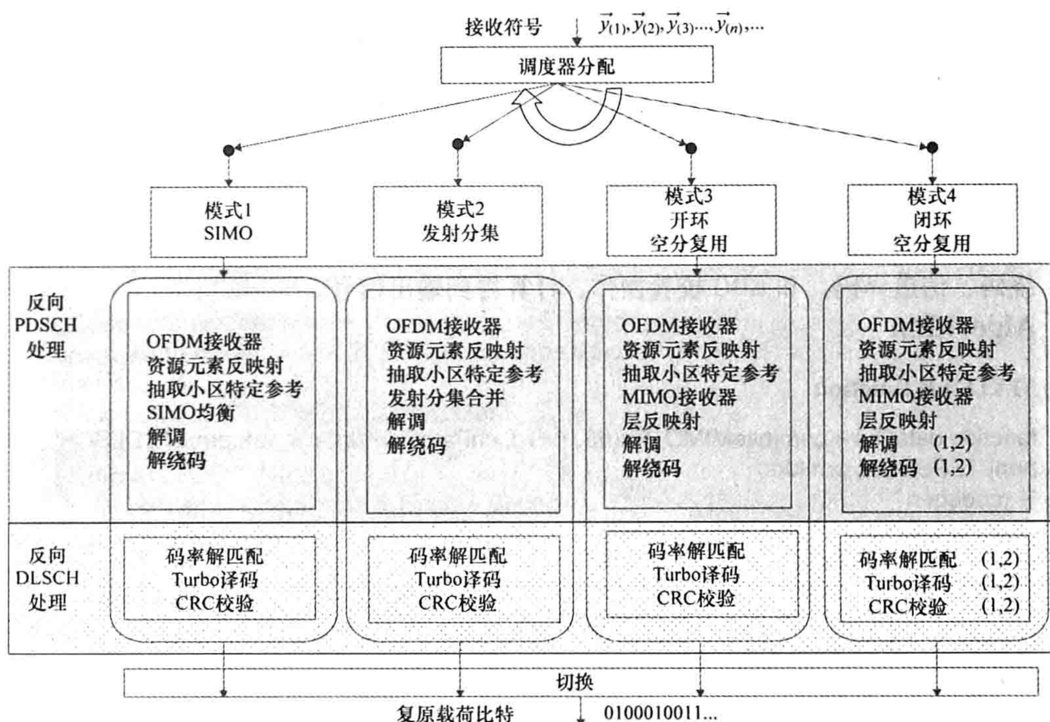


图 8.4 LTE 下行链路系统模型：传输模式 1～4，接收端操作

估计。

在这里，我们基于调度分配的传输模式进行 MIMO 检测操作，以复原调制符号的最优估计。在 SIMO 模式，接收端检测即频域均衡。在发射分集模式，使用发射分集合并器。在空分复用模式，MIMO 接收器对每个信道估计矩阵得到的接收符号求解 MIMO 方程，然后通过层反映射操作，将不同的子比特流分别映射回信号调制流。

开环（模式 3）和闭环（模式 4）MIMO 接收器的不同之处在于预编码矩阵。开环算法对不同的预检测接收符号分配不同的预编码器矩阵，而闭环算法对所有接收符号使用相同的预编码器矩阵。在我们得到调制符号的最优估计后，我们进行解调、解绕码、信道译码，和 CRC 校验操作，以得到载荷比特的最优估计。因球形译码器为 MIMO 接收器算法的一部分，我们跳过包含球形译码器算法的解调操作。MIMO 检测之后的操作重复进行遍历所有发射码字。

8.1.6 接收端模型的 MATLAB 模型

下面的 MATLAB 函数为某一子帧给定传输模式的 LTE 下行链路接收端操作。

函数输入为子帧数（nS）、信道处理得到的 OFDM 信号（rxSig）、每个接收信道的噪声方差估计值（nVar）、信道路径增益矩阵（chPathG）、发射的小区特征参考信号（csr _ ref），以及三个参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。函数输出传输块载荷比特的最优估计（dataOut）。如上一节所属，函数首先进行通用 OFDM 接收和反映射操作复原资源网格，并进行信道估计。在 MATLAB switch - case 声明中，根据所选的传输模式（对应 prmLTEPDSCH. txMode 变量）进行相应的 MIMO 接收器操作。最后，进行通用的解调、解绕码、信道译码，和 CRC 校验操作，计算得到输出信号。

Algorithm

MATLAB function

```
function dataOut = commlteMIMO_Rx(nS, rxSig, chPathG, nVar, csr_ref, prmLTEDLSCH,
prmLTEPDSCH, prmMdl)
%#codegen
% OFDM Rx
rxGrid = OFDMRx(rxSig, prmLTEPDSCH);
% updated for numLayers -> numTx
[dataRx, csrRx, idx_data] = REdemapper_mTx(rxGrid, nS, prmLTEPDSCH);
% MIMO channel estimation
if prmMdl.chEstOn
   chEst = ChanEstimate_mTx(prmLTEPDSCH, csrRx,  csr_ref, prmMdl.chEstOn);
   hD     = ExtChResponse(chEst, idx_data, prmLTEPDSCH);
else
   idealChEst = IdChEst(prmLTEPDSCH, prmMdl, chPathG);
   hD =  ExtChResponse(idealChEst, idx_data, prmLTEPDSCH);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MIMO Receiver based on the mode
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dataOut=false(size(dataIn));
switch prmLTEPDSCH.txMode
   case 1
      % Based on Maximum-Combining Ratio (MCR)
      yRec = Equalizer_simo(dataRx, hD, max(nVar), prmLTEPDSCH.Eqmode);
      cwOut = yRec;

   case 2
      % Based on Transmit Diversity  with SFBC combiner
      yRec = TDCombine(dataRx, hD, prmLTEPDSCH.numTx, prmLTEPDSCH.numRx);
      cwOut = yRec;
   case 3
      yRec = MIMOReceiver_OpenLoop(dataRx, hD, prmLTEPDSCH, nVar);
      % Demap received codeword(s)
      [cwOut, ~] = LayerDemapper(yRec, prmLTEPDSCH);
```

```
    case 4
       % Based on Spatial Multiplexing
       yRec = MIMOReceiver(dataRx, hD, prmLTEPDSCH, nVar, Wn);
       % Demap received codeword(s)
       [cwOut1, cwOut2] = LayerDemapper(yRec, prmLTEPDSCH);
       if  prmLTEPDSCH.numCodeWords==1
          cwOut = cwOut1;
       else
          cwOut = [cwOut1, cwOut2];
       end
end
% Codeword processing
Len=numel(dataOut)/prmLTEPDSCH.numCodeWords;
index=1:Len;
for n = 1:prmLTEPDSCH.numCodeWords
   % Demodulation
   if prmLTEPDSCH.Eqmode == 3
      % not necessary in case of Sphere Decoding
      demodOut = cwOut(:,n);
   else
      % Demodulate
      demodOut = DemodulatorSoft(cwOut(:,n), prmLTEPDSCH.modType, max(nVar));
   end
   % Descramble received codeword
   rxCW =  Descramble(demodOut, nS, 0, prmLTEPDSCH.maxG);
   % Channel decoding includes CB segmentation, turbo decoding, rate dematching
   [decTbData, ~,~] = TbChannelDecoding(nS, rxCW, Kplus(n), C(n),  prmLT-
EDLSCH, prmLTEPDSCH);
   % Transport block CRC detection
   [dataOut(index), ~] = CRCdetector(decTbData);
   index = index +Len;
end
```

8.2 用 MATLAB 构建的系统模型

在本节中，我们创建 LTE 标准 PHY 系统模型的 MATLAB 测试脚本（commlteSystem）。首先脚本调用初始化函数（commlteSystem _ initialize）设置所有相关参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）。随后通过 While 循环调用 MIMO 收发端函数，包括发射端（commlteSystem _ Tx）、信道模型（commlteSystem _ Channel）、和接收端（commlteSystem _ Rx），处理所有子帧。随后，更新比特误码率（BER）并调用可视化函数显示信道响应和均衡前后的调制星座图。通过比较收发比特，我们可以随后进行各种性能评估测量。

Algorithm

MATLAB function

```
% Script for LTE (mode 1 to 4, downlink transmission)
%
% Single or double codeword transmission for mode 4
%
clear functions
%%
commlteSystem_params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteSystem_initialize(txMode, ...
   chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, Doppler, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl Doppler corrLvl chEstOn numCodeWords enPMIfback cbIdx
%%
disp('Simulating the LTE Downlink - Modes 1 to 4');
zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
hPBer = comm.ErrorRate;
%% Simulation loop
tic;
SubFrame =0;
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (Measures(3) < maxNumBits) && (Measures(2) < maxNumErrs)
   %% Transmitter
   [txSig, csr, dataIn] = commlteSystem_Tx(nS, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
    %% Channel model
   [rxSig, chPathG, ~] = commlteSystem_Channel(txSig, snrdB, prmLTEPDSCH,  prmMdl );
    %% Receiver
    nVar=(10.^(0.1.*(-snrdB)))*ones(1,size(rxSig,2));
    [dataOut, dataRx, yRec] = commlteSystem_Rx(nS, csr, rxSig, chPathG, nVar, ...
       prmLTEDLSCH, prmLTEPDSCH, prmMdl);
   %% Calculate  bit errors
   Measures = step(hPBer, dataIn, dataOut);
   %% Visualize results
   if (visualsOn && prmLTEPDSCH.Eqmode~=3)
      zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
   end;
   fprintf(1,'Subframe no. %4d ; BER = %g \r', SubFrame, Measures(1));
   %% Update subframe number
   nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
   SubFrame =SubFrame +1;
end
toc;
```

8.3 定量评估

在本节中，我们以各种角度进行性能评估。通过代入不同仿真参数执行 MATLAB 系统模型，我们可以评估 LTE 标准的性能。首先我们从传输模式角度考察性能。随后，针对给定的一种传输模式，我们改变信道模型考察性能。接下来，我们考察 BER 以确定合适的 MIMO - OFDM 均衡器配置。然后我们验证链路延迟扩散和 OFDM 传输循环前缀（CP）如何影响整体性能。最后，我们考察接收端操作，包括信道估计和 MIMO 接收器算法，对整体性能的影响。

8.3.1 传输模式的影响

在本节中，我们考察 BER 性能和传输模式的关系。我们遍历 9 种测试状态，一种测试状态对应一种传输模式和一个有效天线配置。如在 SIMO 模式（模式 1），我们考察三种有效天线配置（1×1、1×2 和 1×4）。对发射分集（模式 2）和空分复用（模式 3 和模式 4），我们只考察 2×2 和 4×4 天线配置。程序分配通用参数设置发射端和接收端，并分别针对低损耗信道和高损耗信道各执行一次。

Algorithm

MATLAB scripts

```
clear all
TestCases=[...
   1,1,1;
   1,1,2;
   1,1,4;
   2,2,2;
   2,4,4;
   3,2,2;
   3,4,4;
   4,2,2;
   4,4,4];
NumCases=size(TestCases,1);
Ber_vec_Experiment1=zeros(NumCases,1);
for Experiment = 1:NumCases
  txMode        = TestCases(Experiment,1);
   % Transmisson mode one of {1, 2, 3, 4}
  numTx         = TestCases(Experiment,2);
   % Number of transmit antennas
  numRx         = TestCases(Experiment,3);
   % Number of receive antennas
   copyfile('commlteSystem_params
   _distorted.m','commlteSystem_params.m');
   commlteSystemModel;
   Ber_vec_Experiment1(Experiment)=
   Measures(1);
end
```

```
clear all
TestCases=[...
   1,1,1;
   1,1,2;
   1,1,4;
   2,2,2;
   2,4,4;
   3,2,2;
   3,4,4;
   4,2,2;
   4,4,4];
NumCases=size(TestCases,1);
Ber_vec_Experiment2=zeros(NumCases,1);
for Experiment = 1:NumCases
  txMode        = TestCases(Experiment,1);
   % Transmisson mode one of {1, 2, 3, 4}
  numTx         = TestCases(Experiment,2);
   % Number of transmit antennas
  numRx         = TestCases(Experiment,3);
   % Number of receive antennas
copyfile('commlteSystem_params_clean.m',
   'commlteSystem_params.m');
   commlteSystemModel;
   Ber_vec_Experiment2(Experiment)=
   Measures(1);
end
```

通用发射端和接收端参数有很多选择，在本例中我们选择以下参数：16QAM 调制方案、1/2 码率 Turbo 编码、10MHz 带宽、每个子帧两个 PDCCH（物理下行链路控制信道）符号，和单码字。接收端参数包括：最大四次迭代 Turbo 译码并附带早期终止机制，MMSE（对小均方误差）MIMO 接收器。

损耗信道为平坦衰落附加最大 70Hz 多普勒频移，高空间相关性天线、SNR 值为 5dB。表 8.1 所示为不同传输模式和天线配置在劣质信道条件下的 BER 和数据速率测量结果。优质信道条件由频率选择性信道模型、低空间相关性天线配置、零多普勒频移、15dB SNR 表征。表 8.2 为不同传输模式和天线配置在优质信道条件下的 BER 和数据速率测量结果。

表 8.1 传输模式与 BER 性能和数据速率的关系：高损耗信道情况

性能对比结果	天线配置	数据速率/(Mbit/s)	BER
模式 1	1×1	13.88	0.2123
	1×2	13.88	0.0098
	1×4	13.88	0.0004
模式 2	2×2	12.96	0.0075
	4×4	12.81	0.0013
模式 3	2×2	25.46	0.3392
	4×4	50.46	0.4067
模式 4	2×2	25.46	0.2621
	4×4	50.46	0.4167

表 8.2 传输模式与 BER 性能和数据速率的关系：低损耗信道情况

性能对比结果	天线配置	数据速率/(Mbit/s)	BER
模式 1	1×1	13.88	0.0032
	1×2	13.88	$4.2e^{-05}$
	1×4	13.88	0.0
模式 2	2×2	12.96	0.0
	4×4	12.81	0.0
模式 3	2×2	25.46	0.1341
	4×4	50.46	0.2748
模式 4	2×2	25.46	0.0967
	4×4	50.46	0.1948

基于如上结果，我们可以得到如下结论：

1）不论何种传输模式，优质信道条件的结果都好于劣质信道条件。

2）在 SIMO 模式，采用多接收天线分集技术可以提升性能。BER 与最大比

合并（MRC）得到的预期相吻合。

3）发射分集可得到与接收分集相当的性能提升。文献［1］中 TD 性能的理论极限与我们的结果相吻合。

4）在空分复用（SM）模式 3 和模式 4，对两种信道条件，其性能都较低。SM 模式可实现最高的数据速率，其问题在于选取何种参数设定可得到可接受的 BER。这将是下一节我们要解答的问题。

8.3.2 BER 与 SNR 的函数关系

在本节中，我们将找到一个“标尺”验证我们 LTE PHY 仿真器的性能。特别是，我们要明确我们的 SM 模式结果是否满足 LTE 的最低要求。LTE 标准是一种 MIMO－OFDM 系统。它可以通过使用频域均衡，克服多径衰落效应及其带来的码间串扰。在发射端系统模型中，MIMO 和 MIMO 组合操作在编码和绕码操作之后进行。在接收端，通过先反向 OFDM 和 MIMO 操作，我们复原接收的调制子比特流并通过 MIMO 接收器得到调制比特流的最优估计。

这意味着通过观察 MIMO 检测之前的接收信号星座图（见图 8.5），我们可

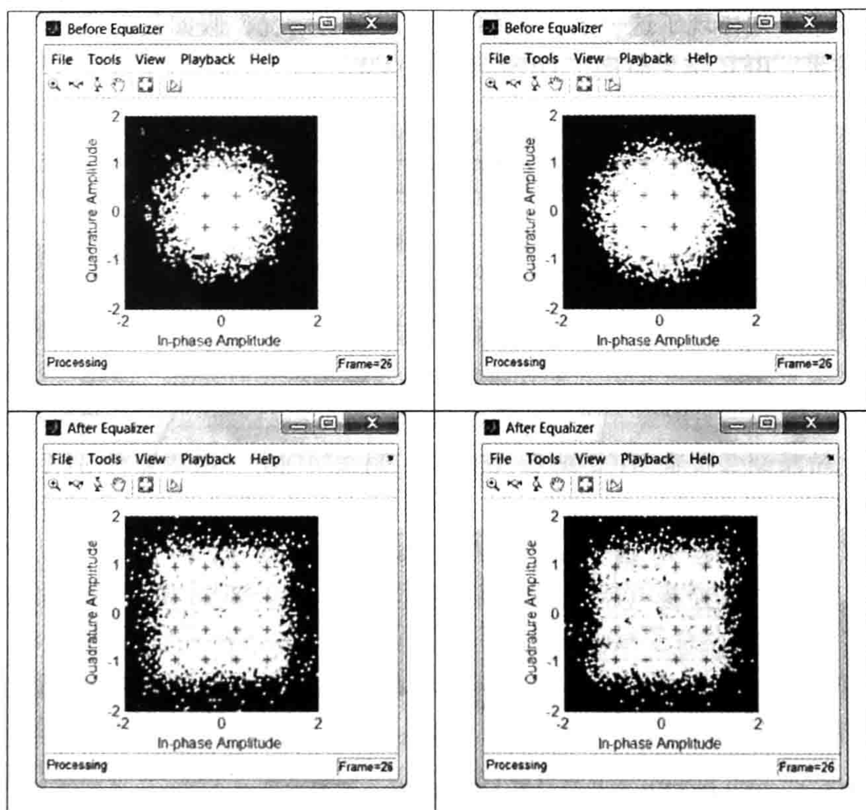


图 8.5 接收信号的星座图：MIMO 检测前后

以快速知道多径衰落造成的影响——即每个调制符号星座点的旋转和衰减。假如 MIMO 和 OFDM 操作如我们预期正确执行，则 MIMO 检测器可以反转和补偿衰落信道造成的劣化。一个优质的 MIMO 检测器包括了一个信道自感知平衡器，它首先计算信道响应随后对每个调制符号的旋转和衰减做出均衡对策。在进行有效的均衡之后，MIMO 检测之后的星座图应与发射符号附加加性高斯噪声的结果类似。因此，尽管均衡之后的信道包含有多径衰落成分，但其依然可以近似等效为一个 AWGN 信道。

在第 4 章中，我们讲解了可以用 BER 曲线表征 Turbo 编码和调制之后的符号在 AWGN 信道传输特性。其 BER 曲线在大于截止 SNR 值时快速下降。因 LTE 系统中一个优质的 MIMO 检测等效信道为一个 AWGN 信道，其系统的 BER 曲线与第 4 章所述相同。

图 8.6 所示为执行系统仿真模型后得到的 BER 曲线随 SNR 值变化的关系。模型传输模式为模式 4，使用频率选择性信道和上一节定义的低损耗信道模型。注意其 BER 曲线形状与 AWGN 信道结果相同。这意味着 MIMO 检测中的频域均衡有效的补偿了多径衰落效应。如图 8.6 中所示，我们可以看到 QPSK 和 16QAM 的结果非常好地体现了这一点。在 64QAM 调制的情况，假如我们选取相近的信道估计技术，BER 结果略逊于 QPSK 和 16QAM。

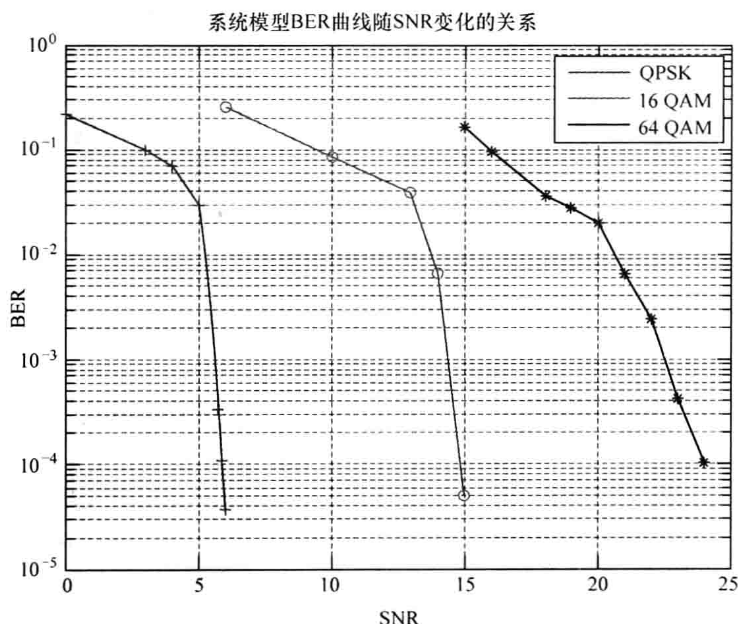


图 8.6 BER 随 SNR 变化的关系——LTE 仿真模型在频率选择性信道条件下：QPSK（左）、16QAM（中）、64QAM（右）

8.3.3 信道估计技术的影响

在本节中，我们考察 BER 性能与信道估计方法之间的关系。其结果与对比传输模式的结果类似。比如，我们选择传输模式 4 和频率选择性信道，在低损耗信道模型设置下运行系统模型。程序通过改变 MATLAB 脚本 commlteSystem _ params 中的 prmMdl. chEst 参数，遍历四种信道估计技术。表 8.3 总结了程序运行的结果。如我们预期，理想信道估计器可以得到最好的 BER 性能。基于内插的估计技术考虑时隙和子帧的最大变化，因而可通过 MMSE 误差最小化方法得到较好的 BER 性能。平均法对提高 BER 性能没有贡献，但其可为更好进行信道感知提供连续性和平滑性。我们将在下一节验证这一影响，我们将用实际音声信号进行仿真并评估声音质量。

表 8.3 BER 性能与信道估计技术的关系

信道估计方法	BER
理想估计	0.0001
内插法	0.0056
时隙平均值法	0.0076
子帧平均值法	0.0086

8.3.4 信道模型的影响

在本节中，我们考察 BER 性能与信道模型之间的关系。其结果与对比传输模式的结果类似。我们选择传输模式 2（发射分集）、2×2 天线配置运行系统模型遍历不同的信道模型。信道模型包括用户定义平坦和频率选择性模型，以及所有 3GPP LTE 信道模型。我们同时通过附加 0、5 或 70Hz 多普勒频移的方式，考虑不同的移动率情况及其对应的空间相关性。SNR 值恒定 13dB。调制方案为 16QAM。程序通过改 chanMdl 参数结构体内最大多普勒频移参数（chanMdl. Doppler）以及空间相关性（chanMdl. corrLvl）参数进行仿真。它们在 MATLAB 脚本 commlteSystem _ params 之中。表 8.4 为结果总结。

随着我们增大噪声等级（用 EPA、扩展步行者 A，和 EVA，扩展车载 A 表征），BER 性能持续下降。高移动率造成性能劣化。同时，随着空间相关性增大，信道矩阵秩不足增大也造成了性能劣化。

表 8.4 BER 性能与信道模型的关系

性能对比结果	最大多普勒频移/Hz	空间相关性	BER
平坦衰落	0	低	0.0
	5	中	$1.3821e^{-02}$
	70	高	$1.1538e^{-02}$
频率选择性衰落（用户自定义）	0	低	0.0
	5	中	$8.0994e^{-06}$
	70	高	$3.4419e^{-03}$
EPA	5	低	0.0
	5	中	$1.5399e^{-03}$
	5	高	$6.6134e^{-03}$
EVA/5Hz	5	低	0.0
	5	中	$4.6661e^{-07}$
	5	高	$2.0997e^{-06}$
EVA/70Hz	70	低	0.0
	70	中	$1.1854e^{-07}$
	70	高	$7.0629e^{-04}$

8.3.5 信道时延扩散与循环前缀的影响

如前文讨论，CP 长度是 OFDM 传输中重要的参数。假如信道模型的路径迟延值大于 CP 长度，则 OFDM 传输无法维持子载波间的正交性。这将导致 BER 性能的劣化。

LTE 标准规定调度器可同时支持普通 CP 和扩展 CP 长度。在高迟延扩散环境中，我们可以切换选择使用扩展 CP 长度以提高性能。

如表 8.5 所示，普通 CP 长度对应所有传输带宽的随大迟延扩散值大约为 4.6875μs。当我们采用用户定义的频率选择性信道模型，并根据 CP 长度设定不同的路径迟延时，BER 性能会受到严重影响。我们采取规定用户定义频率选择性信道模型路径迟延值的方法运行仿真。我们从 0 到最大迟延值等间隔取值作为路径迟延值，如下代码所示。

表 8.5 CP 长度与带宽的对应关系

信道带宽	资源块数量	最小 CP 长度（样本）	信道采样速率/MHz	最大迟延/μs
1.4	6	9	1.92	4.6875
3	15	18	3.84	4.6875
5	25	36	7.68	4.6875
10	50	72	15.36	4.6875
15	75	108	23.04	4.6875
20	100	144	30.72	4.6875

Algorithm

MATLAB code segment in the prmsMdl function

```
% Channel parameters
prmMdl.PathDelays = floor(linespace(0,DelaySpread,5))*(1/chanSRate);
```

在第一种情况中，迟延扩散值在 CP 长度范围内。下面的 MATLAB 代码初始化迟延扩散值小于 CP 长度值。

Algorithm

MATLAB code segment in initialization function – Low delay spread length

```
% Channel parameters
chanSRate   = prmLTEPDSCH.chanSRate;
DelaySpread = prmLTEPDSCH.cpLenR - 2;
 prmMdl = prmsMdl(chanSRate,  DelaySpread, chanMdl, Doppler, numTx, numRx, ...
   corrLvl, chEstOn, enPMIfback, cbIdx);
```

在第二种情况中，迟延扩散值在 CP 长度范围外。下面的 MATLAB 代码初始化迟延扩散值为 CP 长度值的两倍。

Algorithm

MATLAB code segment in initialization function – High delay spread length

```
% Channel parameters
chanSRate   = prmLTEPDSCH.chanSRate;
DelaySpread =  2* prmLTEPDSCH.cpLenR;
 prmMdl = prmsMdl(chanSRate,  DelaySpread, chanMdl, Doppler, numTx, numRx, ...
   corrLvl, chEstOn, enPMIfback, cbIdx);
```

仿真结果与对比传输模式的结果类似。我们采用传输模式 1，1×2 天线配置。SNR 恒为 15dB，16QAM 调制。表 8.6 总结了仿真结果。我们可以看到，迟延扩散即使偶尔超出 CP 长度也会造成严重的性能劣化。

表 8.6 延迟扩散范围对 BER 性能的影响

迟延扩散值	BER
低	0.00019
高	0.02440

8.3.6 MIMO 接收器算法的影响

在本节中，我们考察 BER 性能与 MIMO 接收器算法的关系。通过改变均衡器模式（prmLTEPDSCH 中 Eqmode 参数）值 1、2、3，我们可以分别选择迫零

(ZF)、MMSE，和球形译码器算法。表 8.7 为仿真结果。

ZF 算法忽略噪声功率最为简单易行，但它的 BER 性能最差。MMSE 算法的 BER 性能好与 ZF 性能。MMSE 算法求反信道矩阵并考虑噪声功率。不过，球形译码器的性能显然最优，它使用最大似然译码方法基于符号映射优化了调制符号。球形译码器的计算复杂度相对较高，它的处理时间比 MMSE 长很多。因此，MIMO 接收器选择 MMSE 或球形译码器需要权衡复杂度和性能的这对折中。

表 8.7　MIMO 检测算法对 BER 性能的影响

MIMO 检测方法	BER
ZF 算法	0.0001
MMSE 算法	0.0056
软判决球形译码	0.0076

8.4　吞吐量分析

LTE 标准不仅仅定义了发射端操作，也定义了多种信道条件以测试和定义符合标准的最低性能要求。比如，标准文件 TS 36.101 为下行链路所有最低性能要求。图 8.7 所示为这一文件内容的核心内容。如，SIMO 传输模式的一个吞吐量要求由一系列测试范例给出，范例包括一组给定的输入及其响应的预期输出。输入定义包括带宽、参考信道、传播参数（信道模式）、天线空间相关性矩阵，和参考 SNR 值。吞吐量定义为收发成功情况下的平均数据速率。这种情况下定义的最大吞吐量不考虑错误接收。相对吞吐量为成功接收吞吐量与最大吞吐量的百分比。例如，测试样本 1，对应 QPSK（正交相移键控）调制、传输模式 1、1×2 天线配置、10MHz 带宽、EVA 信道模型（多普勒频移 5Hz)，低空间相关性情况下，－1dB SNR 对应 70% 相对吞吐量预期。

测试号	带宽/MHz	参考信道	OCNG 模式	传播条件	相关性矩阵和天线配置	占最大吞吐量百分比(%)	SNR /dB	UE 类型
1	10	R.2 FDD	OP.1 FDD	EVA5	1×2 Low	70	−1.0	1−8
1A	2×10	R.2 FDD	OP.1 FDD (Note 1)	EVA5	1×2 Low	70	−1.1	3−8
2	10	R.2 FDD	OP.1 FDD	ETU70	1×2 Low	70	−0.4	1−8
3	10	R.2 FDD	OP.1 FDD	ETU300	1×2 Low	70	0.0	1−8
4	10	R.2 FDD	OP.1 FDD	HST	1×2 Low	70	−2.4	1−8
5	1.4	R.4 FDD	OP.1 FDD	EVA5	1×2 Low	70	0.0	1−8
...	...	...	...	...	...	...	...	...

图 8.7　LTE 下行链路测试范例：摘取 TS 36.101 中最低测试要求[2]. 来自 3GPP 文件

我们已经更新了接收端和系统的 MATLAB 函数以计算吞吐量。在接收端，最后一步进行 CRC 校验。当 CRC 校验发现一个错误时，这个错误也已存于与输出中。通过从全部输出中排除错误输出的处理，我们可以通过将错误输出与全部输出相除，算出相对吞吐量。下面的 MATLAB 函数按照定义计算并显示相对吞吐量。

Algorithm

MATLAB function

```
function Throughput=getThroughput( ber, CbFlag, SubFrame)
persistent ErrorBlk
if isempty(ErrorBlk)
   ErrorBlk=0;
end
ErrorBlk = ErrorBlk + CbFlag;
Throughput=1-(ErrorBlk/SubFrame);
fprintf(1,'Subframe %4d ; BER = %6.4f ; ErrorFrame = %4d ; Throughput = %4.2f \r', ...
      SubFrame, ber, ErrorBlk, Throughput );
end
```

8.5 用 Simulink 进行系统建模

到目前为止，我们已经用 MATLAB 算法和测试脚本仿真 LTE 标准的物理层。在本节中，我们会使用 Simulink 设计实现同一个系统模型。Simulink 模型内嵌了仿真测试平台，可以让我只关注算法而不需要维护测试脚本。让我们重新检视 MATLAB 系统模型，将算法部分（如与系统处理有关的代码）从测试部分中挑选出来。

Algorithm

MATLAB function

```
clear functions
commlteSystem_params;
[prmLTEPDSCH, prmLTEDLSCH, prmMdl] = commlteSystem_initialize(txMode, ...
   chanBW, contReg, modType, Eqmode,numTx, numRx,cRate,maxIter, fullDecode,
chanMdl, Doppler, corrLvl, ...
   chEstOn, numCodeWords, enPMIfback, cbIdx);
clear txMode chanBW contReg modType Eqmode numTx numRx cRate maxIter
fullDecode chanMdl Doppler corrLvl chEstOn numCodeWords enPMIfback cbIdx
%%
disp('Simulating the LTE Downlink - Modes 1 to 4');
zReport_data_rate_average(prmLTEPDSCH, prmLTEDLSCH);
hPBer = comm.ErrorRate;
```

```
%% Simulation loop
tic;
SubFrame =0;
nS = 0; % Slot number, one of [0:2:18]
Measures = zeros(3,1); %initialize BER output
while (Measures(3) < maxNumBits) && (Measures(2) < maxNumErrs)
----------------------------------------------------------------------
   %% Transmitter
   [txSig, csr, dataIn] = commlteSystem_Tx(nS, prmLTEDLSCH, prmLTEPDSCH, prmMdl);
    %% Channel model
   [rxSig, chPathG, ~] = commlteSystem_Channel(txSig, snrdB, prmLTEPDSCH,  prmMdl );
    %% Receiver
    nVar=(10.^(0.1.*(-snrdB)))*ones(1,size(rxSig,2));
    [dataOut, dataRx, yRec] = commlteSystem_Rx(nS, csr, rxSig, chPathG, nVar, ...
       prmLTEDLSCH, prmLTEPDSCH, prmMdl);
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
----------------------------------------------------------------------
   %% Calculate  bit errors
   Measures = step(hPBer, dataIn, dataOut);
   %% Visualize results
   if (visualsOn && prmLTEPDSCH.Eqmode~=3)
      zVisualize( prmLTEPDSCH, txSig, rxSig, yRec, dataRx, csr, nS);
   end;
   fprintf(1,'Subframe no. %4d ; BER = %g \r', SubFrame, Measures(1));
   %% Update subframe number
   nS = nS + 2; if nS > 19, nS = mod(nS, 20); end;
   SubFrame =SubFrame +1;
end
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
toc;
```

我们可以将 MATLAB 系统模型分为三部分：

1）初始化：设置各个系统参数。在处理循环启动前运行。

2）调度：一个 While 循环遍历子帧处理。另需器他操作以更新 While 循环条件。

3）循环内处理：子帧处理包括发射端、信道模型，和输出端操作。并有额外代码比较输入输出比特并可视化结果。

当我们在 Simulink 中构建同一个模型时，我们只需对循环内处理部分进行建模。调度部分由 Simulink 自动完成。Simulink 通过时基仿真引擎遍历每个数据采样或帧直到满足规定仿真时间或终止条件。因为 MATLAB 和 Simulink 共享数据平台，我们可以在 Simulink 仿真之前手动进行初始化，或在打开模型或仿真开始之前设定 Simulink 模型初始化代码。

8.5.1 构建一个 Simulink 模型

我们可以从 Simulink 库中选取组件构建 LTE 收发系统的 Simulink 模型。通过在 MATLAB 环境中点击 Simulink 库图标可以方便地访问 Simulink 库，如图 8.8 所示。在 Simulink 库浏览器中，有各种 Simulink 和其他 MathWorks 产品组件集合。我们最常用的 Simulink 组件为 DSP System Toolbox，和 Communications System Toolbox。如图 8.9 所示，Simulink 库组件调用用户自定义函数。

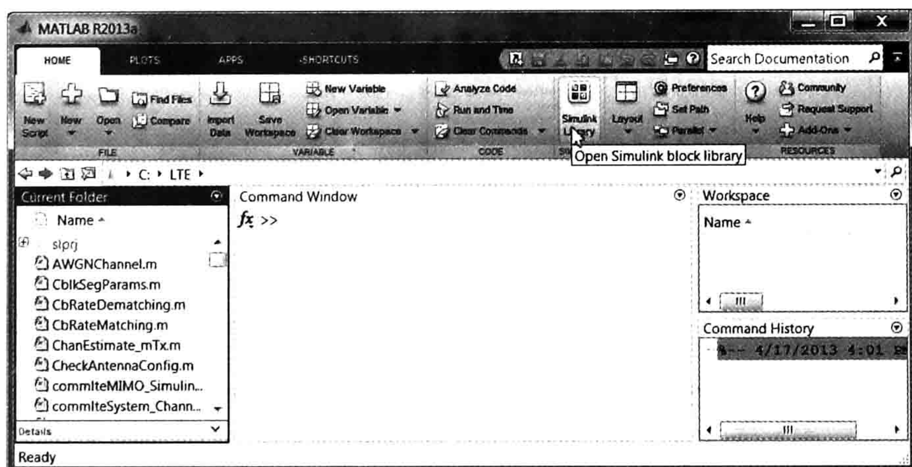


图 8.8 从 MATLAB 环境中访问 Simulink 库

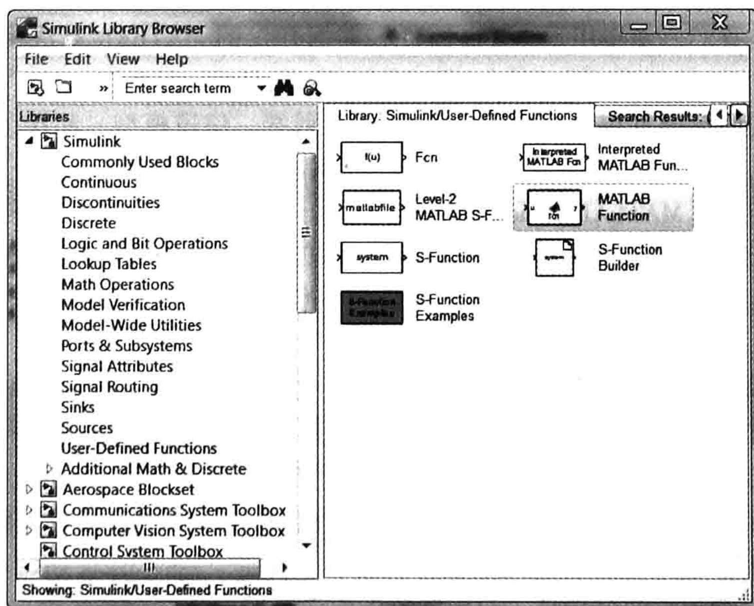


图 8.9 Simulink 组件库，列出了主要的 Simulink 组件和其他产品库

我们通过以下方式建立一个新的 Simulink 模型，在 Simulink 库浏览器中 Menu 菜单中选择：File→New→Model。这样就建立了一个空的 Simulink 模型，如图 8.10 所示。

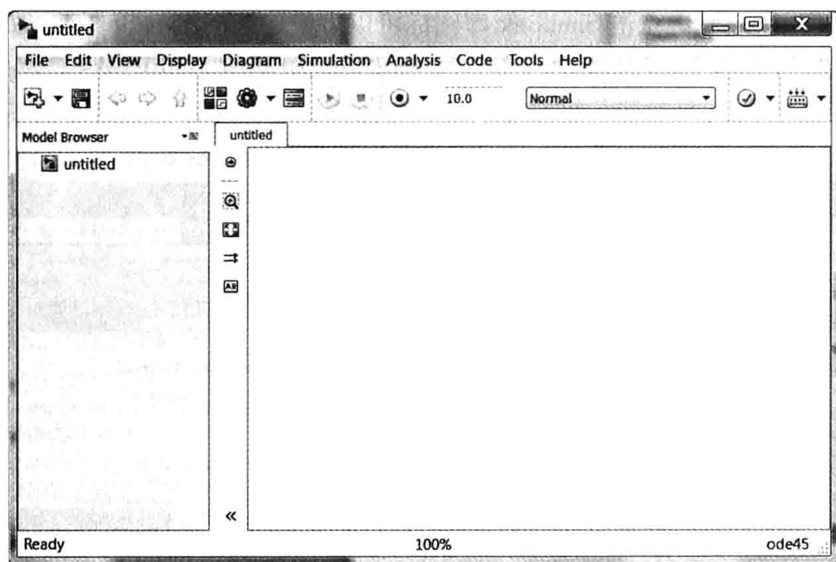


图 8.10　为 LTE 收发端建立一个新的 Simulink 模型：建立空模型

8.5.2　Simulink 集成 MATLAB 算法

下面，我们一步步用 Simulink 组件和前面开发的 MATLAB 算法构建 LTE 收发端模型。

因为我们希望复用 MATLAB 算法构建发射端、信道模型，和接收端，所以我们需要一种组件可以将 MATLAB 函数转换为 Simulink 模型。一个称为 MATLAB 函数组件（MATLAB Function Block）的组件可以完成这项工作，它在 Simulink 用户自定义函数库中。我们需要复制 MATLAB 函数组件（MATLAB Function Block）组件四次。如图 8.11 所示，我们可以更改组件名以标识它的功能：发射端、信道、接收端，和子帧更新。

我们通过单击组件“发射端”图标打开它。当我们打开任意 MATLAB 函数组件（MATLAB Function Block）时，MATLAB 编辑器中会默认生成一个函数定义，如图 8.12 所示。接下来我们可以重新定义和更新这个默认函数，添加进我们需要的内容。当我们添加输入声明时，组件会自动生成输入端口，而输出声明会生成输出端口。

在这里我们有两种选择。我们即可以将发射端函数体（commlteSystem _ Tx）拷贝到函数定义内，也可以让组件函数调用发射端函数，如图 8.13 所示，可将

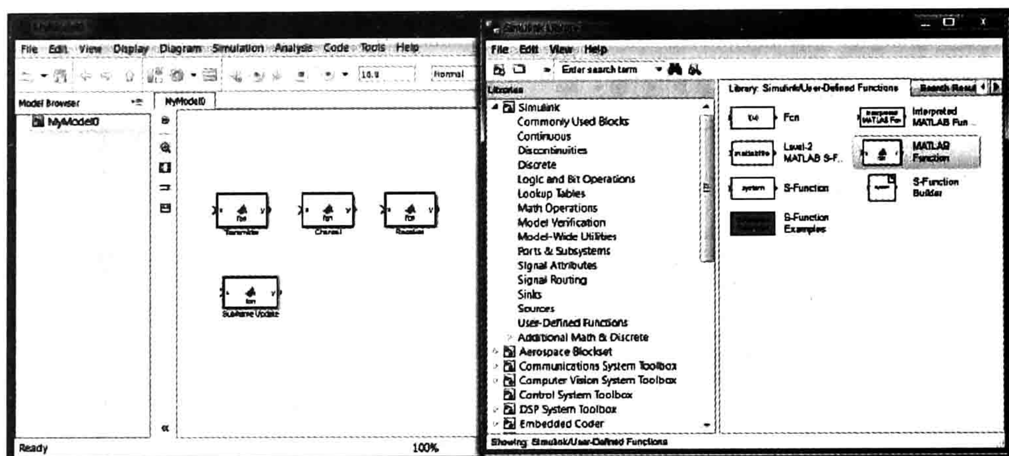


图 8.11 向 Simulink 模型中添加 MATLAB 函数

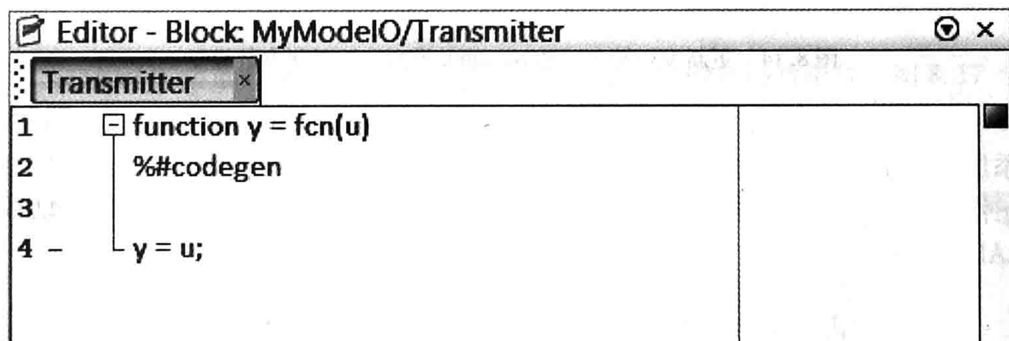


图 8.12 MATLAB 函数组件，有一个默认函数定义

组件函数输入变量和发射端函数输出变量相关联。

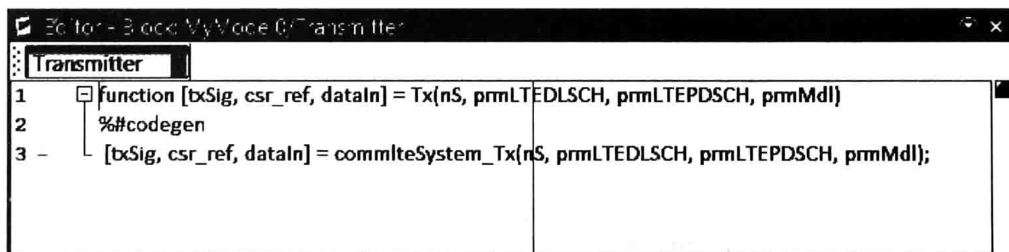


图 8.13 MATLAB 函数组件：更新函数声明配置发射端

在保存这个函数之后，我们点击返回模型界面（Go to Diagram）图标回到模型界面。如图 8.14 所示，我们可以看到发射端组件自动更新并反映了函数定义的变更。在这一步，所有的默认函数声明都对应了相应的输入和输出端口。

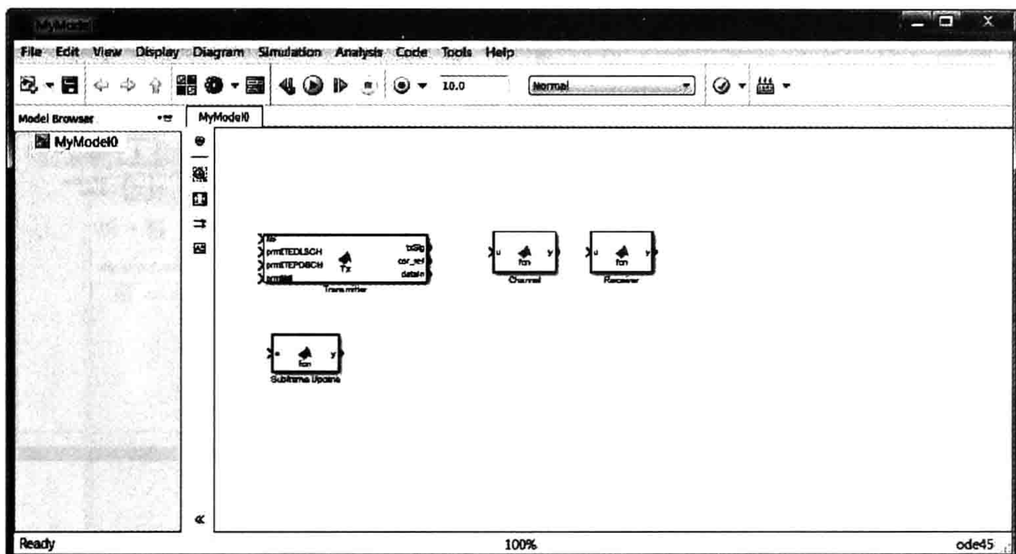


图 8.14　更新发射端之后的 Simulink 模型，无任何参数设定

我们下面将所有相关参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）添加如 Simulink 模型参数中。这些参数结构体在仿真中保持不变，并在 MATLAB 工作区中以三个变量形式被 Simulink 模型访问。我们打开发射端组件并在 MATLAB 编辑器中点击编辑数据（Edit Data）图标，如图 8.15 所示。

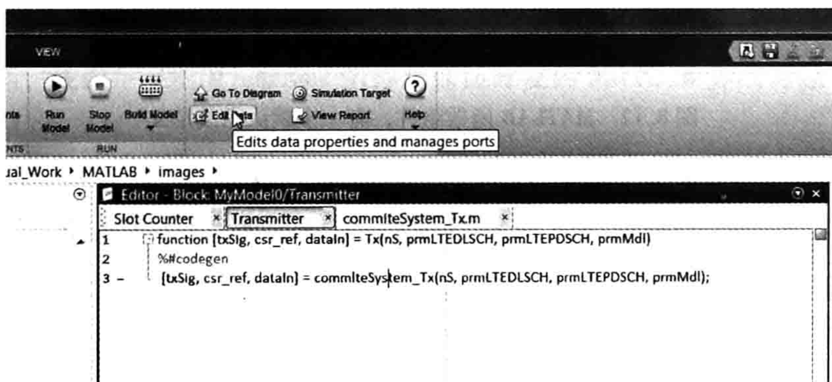


图 8.15　访问端口和数据管理器显示发射端 LTE 模型参数结构体声明

界面中出现一个叫做端口与数据管理器（Port and Data Manager）的对话框。我们可以在其中编辑数据属性并管理端口和参数。这个对话框如图 8.16 所示。接下来，我们点击每个端口名（prmLTEDLSCH，prmLTEPDSCH，prmMdl），变更其 Scope 属性为参数（parameter），将可调（Tunable）选项卡勾销，并点击应用（Apply）。

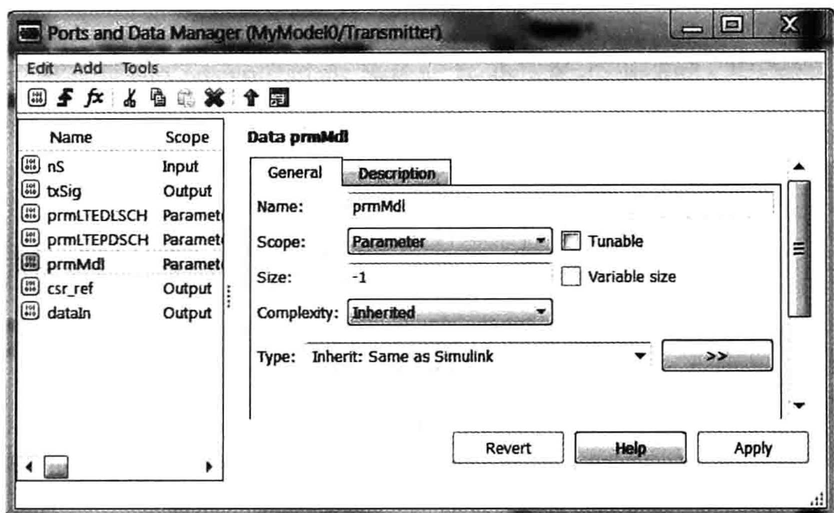


图 8.16 设定 LTE 参数结构体为不可调类型参数

我们接下来重复这一过程更新信道和接收端 MATLAB 函数组件。图 8.17 显示了组件函数调用信道和接收端函数的例子，可以看到前文中的 MATLAB 算法可以直接集成到 Simulink 中。

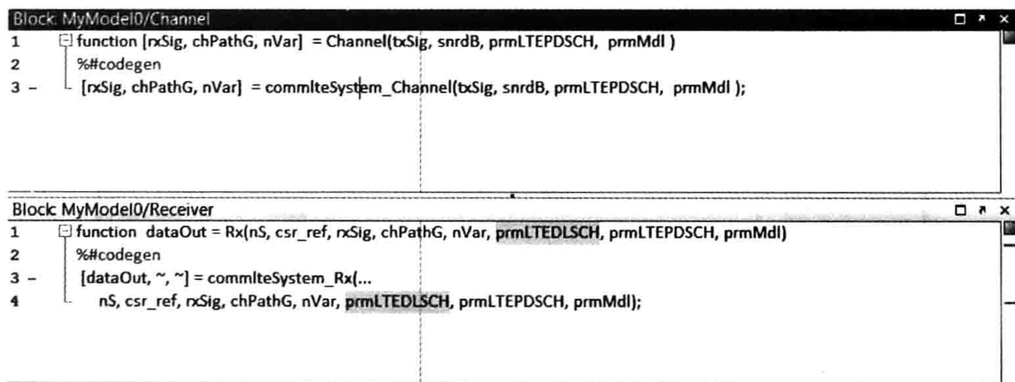


图 8.17 在相应的 MATLAB 函数组件中调用信道模型和接收端函数

在这一步我们可以将发射端组件的输出（txSig）与信道组件的输入相连。我们也可以将信道组件的三个输出信号（rxSig，chPathG，nVar）与接收端组件输入相连，如图 8.18 所示。

现在我们需要更新子帧数（nS）。子帧数是发射端和接收端组件的公共输入。为了避免在模型中添加过多连线，我们使用 Simulink Signal Routing 库中的 GoTo 和 From 组件。子帧更新组件的输出为当前帧的时隙数。我们通过点击组件，为信号分配一个名字——即添加一个标签，将这个输出信号与 GoTo 组件相

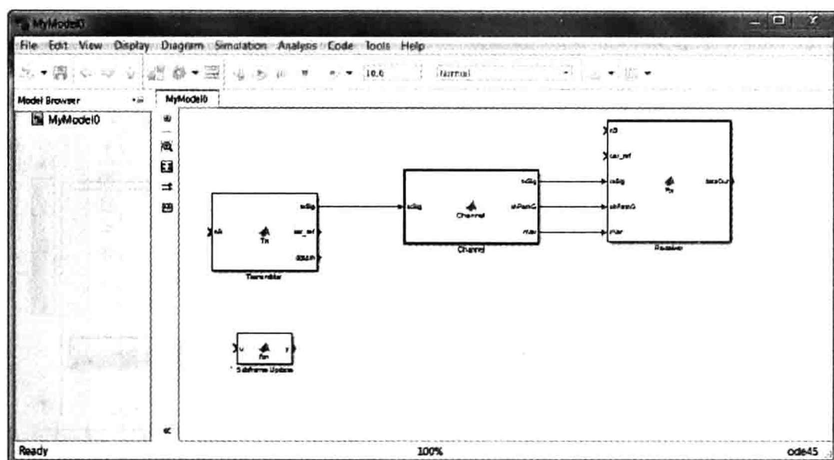


图 8.18　连接发射端、信道模型和接收端组件

连。现在，模型中有相同标签的任意 From 组件都将与这个输出信号相连。如图 8.19 所示，我们用两个 From 组件将子帧数输出发射端和接收端组件。

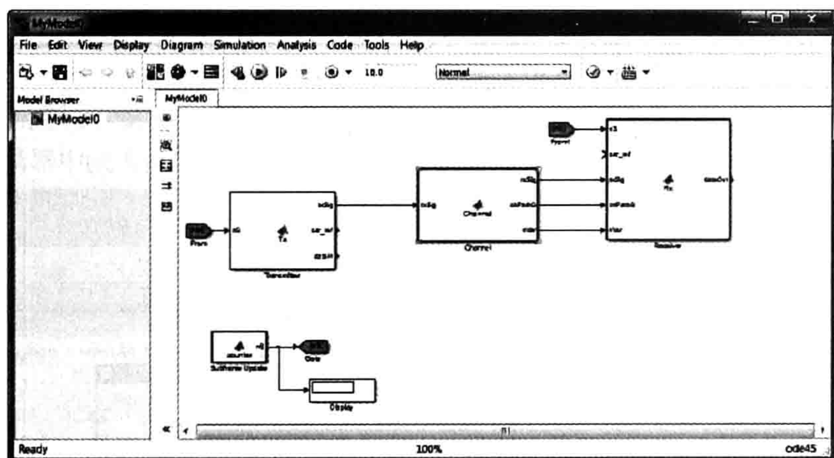


图 8.19　使用 GoTo 和 From 组件将子帧数与发射端接收端组件相连

当我们双击子帧更新组件时，我们会看到其 MATLAB 函数如图 8.20 所示。这段代码与我们前文的 MATLAB 测试脚本中更新子帧数代码相同。在子帧更新中，我们用一个递增变量设定每 10ms 帧清零一次计数器。现在我们用 GoTo 和 From 组件完成模型内所有组件的互连。如图 8.21 所示，一对 GoTo 和 From 组件标注了一个标签（csr），背景为紫色。这确保了一个子帧的小区专有信号（导频）同为发射端和接收端所使用。我们同时使用另外两个 GoTo 组件将发射输入比特流（dataIn）和接收端输出比特流（dataOut）集合在一起。在这个 Goto 组

件中，我们用标签 Input 和 Output 分别标注信号 dataIn 和 dataOut，并用同一种背景颜色。我们使用 Communications System Toolbox 的 CommSinks 库中的 Error rate calculation 组件计算系统的 BER。通过用两个标注 Input 和 Output 的 From 组件，我们将发射输入比特流和接收端输出比特流输入 Error Rate Calculation 组件，如图 8.22 所示。Error Rate Calculation 组件在仿真中不断对比每一个子帧的译码比特和源比特计算 BER。这个组件的输出为包含 BER、错误比特数，和处理比特数的三元向量。

Editor - Block: MyModel0/Subframe Update

Subframe Update

```
1     function nS = counter
2     % Assumes two slots per time step, i.e. modeling a subframe per step.
3     %#codegen
4 -   persistent numSlot;        % slot number in radio frame, [0, 19]
5 -   if isempty(numSlot)
6 -       numSlot = 0;
7     else
8 -       numSlot = numSlot+2;    % Update slot number, 2 slots per subframe
9     end
10 -  if numSlot > 19
11 -      numSlot = mod(numSlot, 20);
12    end
13 -  nS = numSlot;
```

图 8.20　子帧更新组件为一个计数器

图 8.21　用 GoTo 和 From 组件连接各组件输入输出端口

模型检查 Error rate calculation 组件的 Stop Simulation 参数以控制仿真时间（见图 8.23）。仿真在检测满足如下两个目标参数中任意一个时终止：最大误码数（通过 maxNumErrs 参数定义）或最大比特数（通过 maxNumBits 参数定义）。

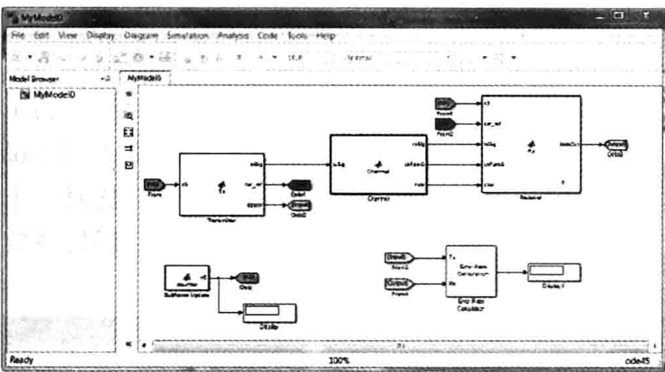


图 8.22 在 Simulink 中完成循环体内处理并计算 BER

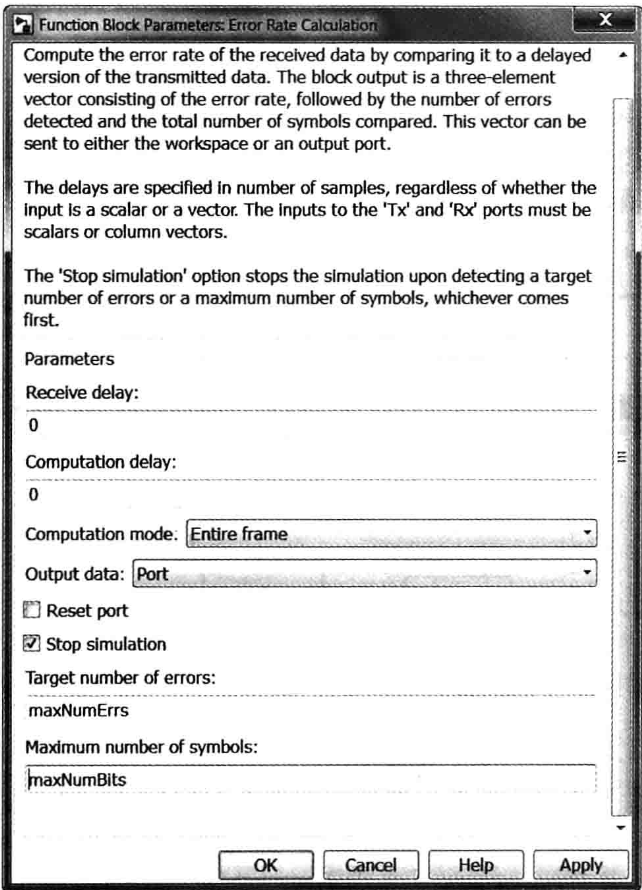


图 8.23 Error rate calculation 组件参数控制仿真时间

这时，我们已经完成 Simulink 模型中所有循环体内处理。接下来我们初始化模型和 LTE 参数结构体并运行 Simulink 模型。Simulink 模型参数可以通过多种途

径初始化，但我们将通过下面两种办法进行：

1）在打开模型时设定模型属性；

2）使用蒙版子系统提供参数设定对话框。

8.5.3 参数初始化

系统模型中的一些操作在仿真循环启动后只需运行一次。这些操作即系统初始化。到目前为止，在 Simulink 模型中所有操作都为循环的一部分并在仿真中重复运行。Simulink 中定义初始化操作的一种方法为使用模型属性（Model Properties），特别是 Callback 函数。

如图 8.24 所示，我们可以在模型的 Menu 菜单中访问模型属性（Model Properties）：File→ Model Properties→ Model Properties.

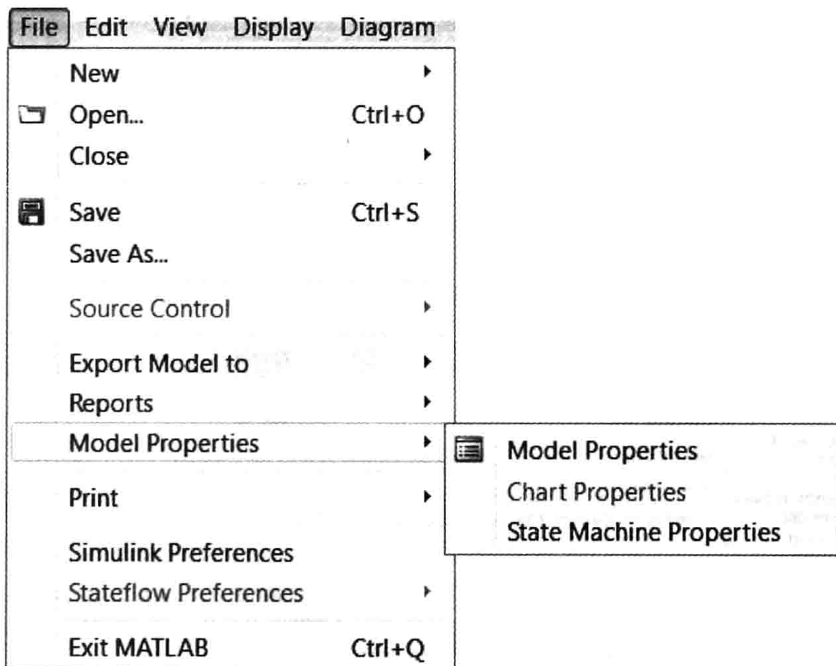


图 8.24 访问 Simulink 模型的模型属性

当我们打开模型属性对话框后，我们可以在 Callback 栏内找到各种模型 Callback。对应每一种类型的模型 Callback 中，我们都可以找到一个可以执行 MATLAB 函数或命令编辑窗口。回传类型与仿真的各阶段有关。例如，在图 8.25 中，我们选择 PreLoadFcn 回传。这表示初始化函数如在我们 MATLAB 模型 While 循环之前执行一样，将在我们载入（或打开）Simulink 模型前执行。

在这里，Simulink 模型即包含了初始化操作也包含了循环内处理。当我们打

开模型和点击 Run 按钮启动模型时，Simulink 执行初始化工作。初始化的最后一步为定义 Simulink 求解器以及，如所需的话，设定仿真采样时间。我们可以通过 Menu 菜单访问 Simulink 求解器：Simulink→Model Configuration Parameters→Solvers。当我们进行通信系统数字基带仿真时，仿真使用离散采样。因此，如图 8.26 所示，在求解器选项属性（Solver Options）中，我们应该选择定步长（Fixed - Step）类型和离散（Discrete，非连续状态）求解器。对大部分 DSP 或无模拟和混合信号的通信系统 Simulink 模型来说，这一设定皆适用。

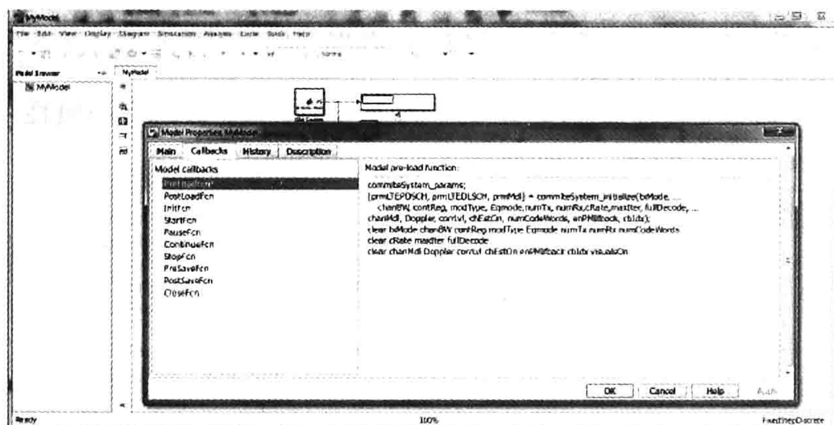


图 8.25　定义初始化命令 PreLoadFcn 回传

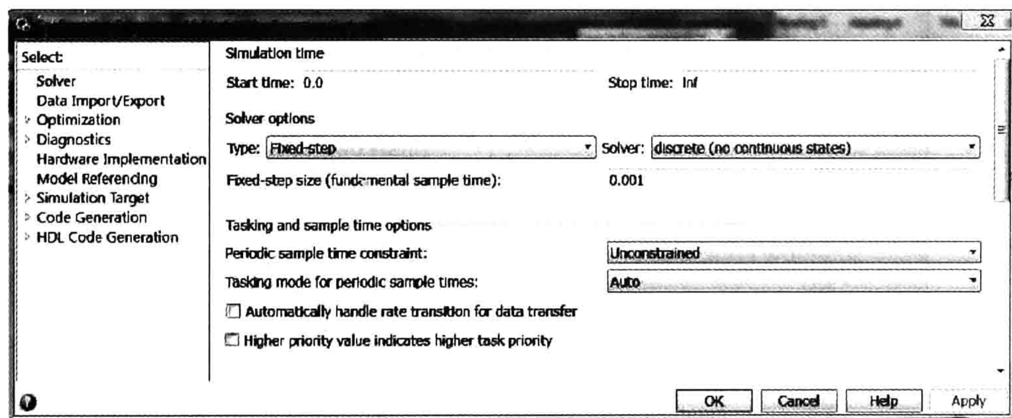


图 8.26　设定求解器选项、采样时间，和仿真模型终止时间

另外，因我们根据 Error rate calculation 组件的参数终止仿真，在求解器对话框中我们可随意对终止时间（Stop Time）赋值。这里我们选择无穷大（通过 inf 参数定义）。不仅如此，因为模型处理单位为 1ms 长度的子帧，所以在 Fixed - Step Size 属性（基础采样时间）对话框内我们键入值 0.001s。

现在我们的 Simulink 模型已定义完成，我们可以运行模型来检测是否初始化 Callbacks 正常工作以及收发端正常工作。

我们保存并关闭模型界面。然后再一次打开它，三个 LTE 参数结构体（prmLTEDLSCH，prmLTEPDSCH，prmMdl）以及三个附加参数（snrdB，maxNumBits，maxNumErrs）自动在 MATLAB 工作区生成。这表明 Callbacks 正常工作，如图 8.27 所示。

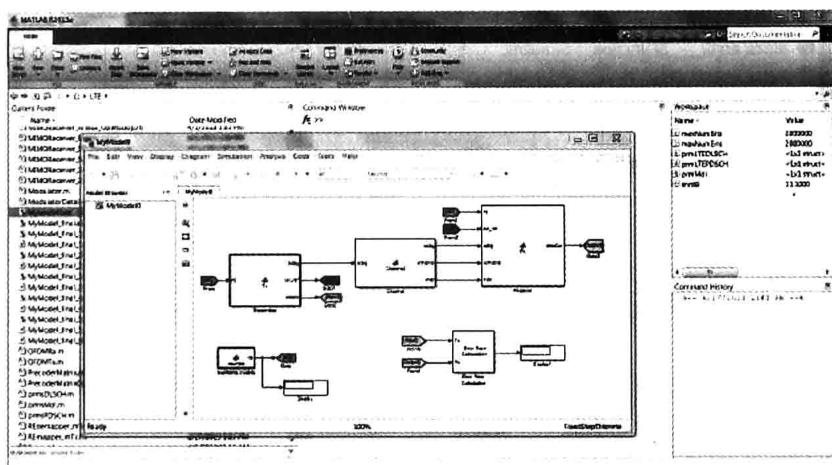


图 8.27 打开模型时 Callbacks 自动生成仿真参数

8.5.4 运行仿真

通过仿真，我们可以测试收发器是否正常工作。我们通过点击 Model 编辑器的 Run 按钮执行仿真。仿真运行时，Simulink 引擎会将模型转换为可执行格式，即模型编译。在编译的第一步，Simulink 引擎根据模型组件参数表达检查一致性、确定所有信号属性并验证每个组件由哪些输入信号。

当模型包括 MATLAB 函数组件时，Simulink 引擎首先将 MATLAB 代码在 MATLAB 函数组件内部转换为 C 代码，随后编译 C 代码生成可供 MATLAB 函数组件执行的格式。假如 MATLAB 函数组件内包含代码生成器不支持的代码，我们就需要改写它们。最后，在连接阶段，为每个信号分配数据存储空间，并连接所有编译过的可执行代码，以完成执行前的准备。

仿真错误信息即可能在编译时出现，也可能在执行时出现。在仿真出现错误的时候，Simulink 会中断仿真，打开导致错误的组件并在 Simulation Diagnostics Viewer 中显示有关错误信息。如图 8.28 中的例子所示，Simulink 引擎注意检查每个组件间端口连接的一致性；Simulink 引擎判断子帧数（nS）信号影响了发射比特和接受比特长度（dataIn 和 dataOut）。因此对每个子帧，收发比特的长度可

能会被改变。在默认情况下，假如我们不再 MATLAB 函数组件中定义任何信号长度，这个信号长度会默认为常数。因此 Simulink 弹出关于这两个信号长度的错误信息“可变长度”，如图 8.28 所示。

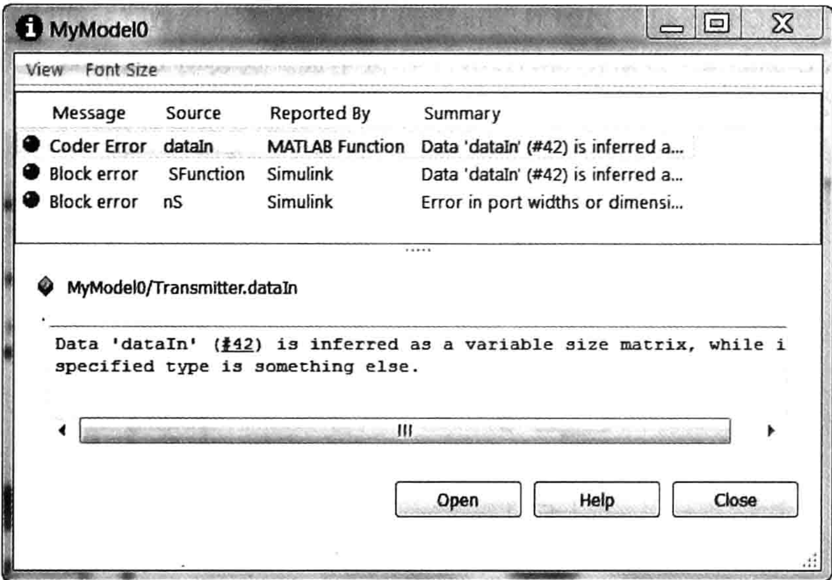


图 8.28 当 Simulink 检查发射信号长度时弹出错误信息

为了解决这个编译错误，我们需要将 dataIn 和 dataOut 这两个信号标记为可变长度信号，并定义其最大长度。我们需要双击发射端和接收端组件，访问它们的端口与数据管理器对话框，如前文所述，选择 dataIn 和 dataOut 这两个信号，通过勾选可变长度改变它们的长度属性并定义其最大长度。这一操作如图 8.29 所示。

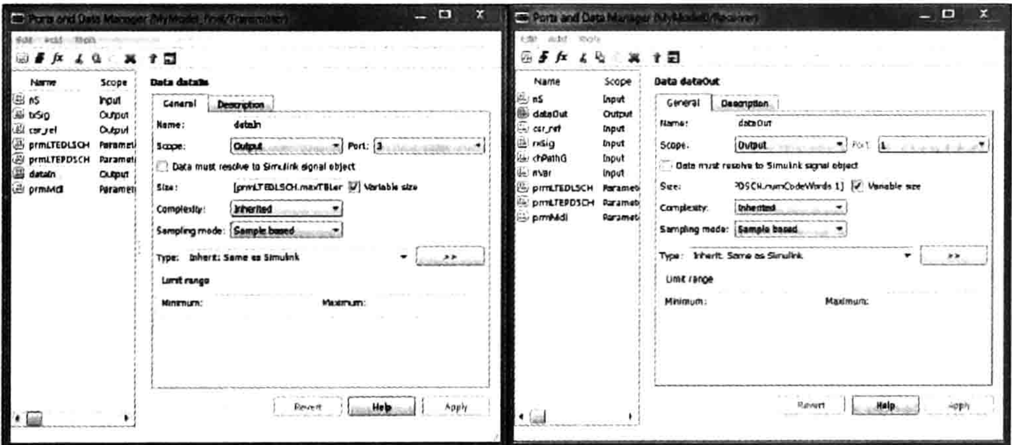


图 8.29 配置发射端和接收端输出信号长度为可变并设置最大长度

可变长度类型信号操作在通信系统 Simulink 模型中经常进行。其原因一部分是因为在仿真中信号经常从占用一个帧变成占用两个帧的长度。

现在我们可以点击 Run 按钮顺利运行仿真。仿真会一直运行直到处理完规定数量的比特。仿真会在 Error rate calculation 组件检测满足终止条件时停止。BER 结果如图 8.30 所示。

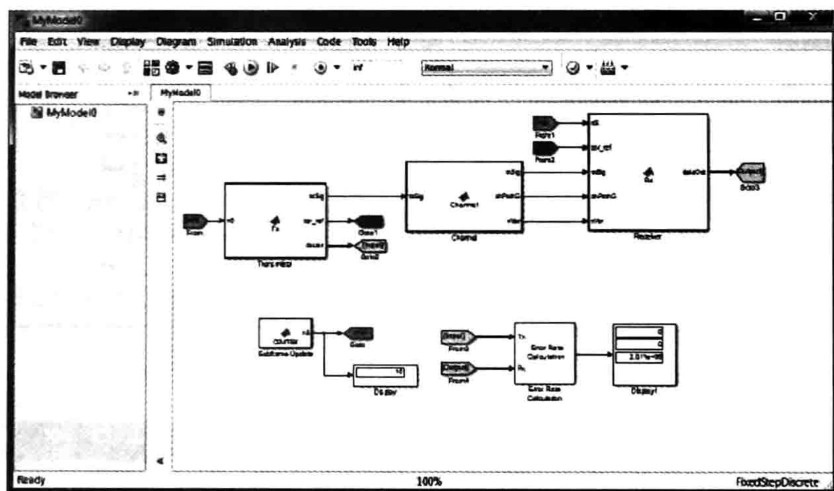


图 8.30　运行 Simulink 系统模型并测量 BER 性能

结果反映了 MATLAB 脚本（commlteSystem _ params）初始化定义的系统参数。假如我们想再不同传输模式或不同条件设定下运行仿真，我们必须首先更改 MATLAB 脚本更新系统参数。然后我们需要返回 Simulink 模型重新运行仿真。当我们运行多个仿真时，这种在 MATLAB 和 Simulink 间反复操作变得十分烦琐。为了简化参数定义，在下一节我们将开发一个 Simulink 参数对话框。

8.5.5 引入参数对话框

参数对话框可以在 Simulink 中更直接和更直观的更新系统参数。实际上，我们需要在我们的 Simulink 模型中引入一个新的子系统，它包含所有模型参数并允许我们很方便的更新它们。这种特殊的子系统即蒙板子系统。

一个蒙板子系统是一个 Simulink 模型中一个或多个组件的集合。每个子系统都可以被遮掩；即有一个设定子系统参数的对话框。不过，在本节中我们引入的子系统是为了囊括和更新所有模型参数。如图 8.31，我们从 Ports and Subsystems Simulink 库中添加子系统组件。

当我们双击 Subsystem 组件图标并打开它时，我们注意到它包含众多输入端口到输出端口的连接。注意在本节中，我们并不需要真正构建一个子系统而是使用这个组件创建蒙板以控制所有系统参数。因此，我们首先从子系统组件中移除

所有输入输出和连接。现在，子系统应该是空的，如图 8.32 所示。为了在我们的模型中将这个子系统组件从其他组件和子系统中区分出来，我们改变其背景色并命名为 Model Parameters，如图 8.33 所示。

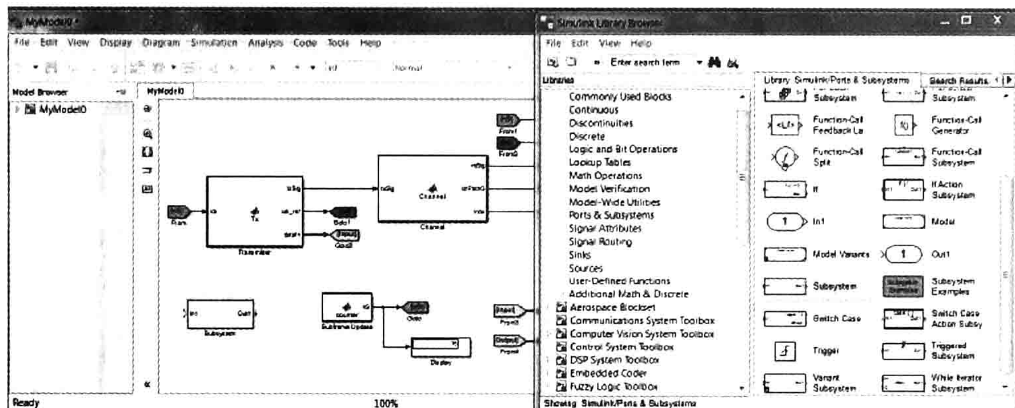


图 8.31　从 Ports and Subsystems Simulink 库中添加 subsystem 组件

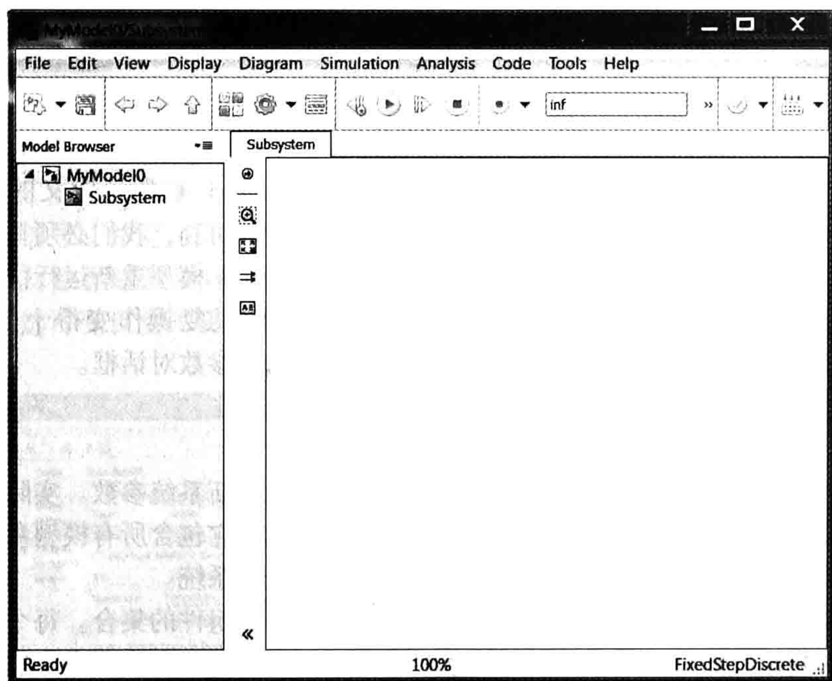


图 8.32　清空子系统以构建参数设置蒙板

下一步，我们将空的子系统转换为蒙板子系统。如图 8.34 所示，首先点击子系统选择它，并打开 Menu 菜单中如下选项：Diagram—> Mask—> Create Mask。这样一个子系统组件就轻松变为一个蒙板。当我们进行这一步骤时，双

击子系统组件图标将不再弹出内容编辑器。取而代之的时一个蒙板编辑器，它可以允许我们添加和定义各种参数并初始化它们。图 8.35 所示为模型参数子系统组件的蒙板编辑器界面，现在它为空白。

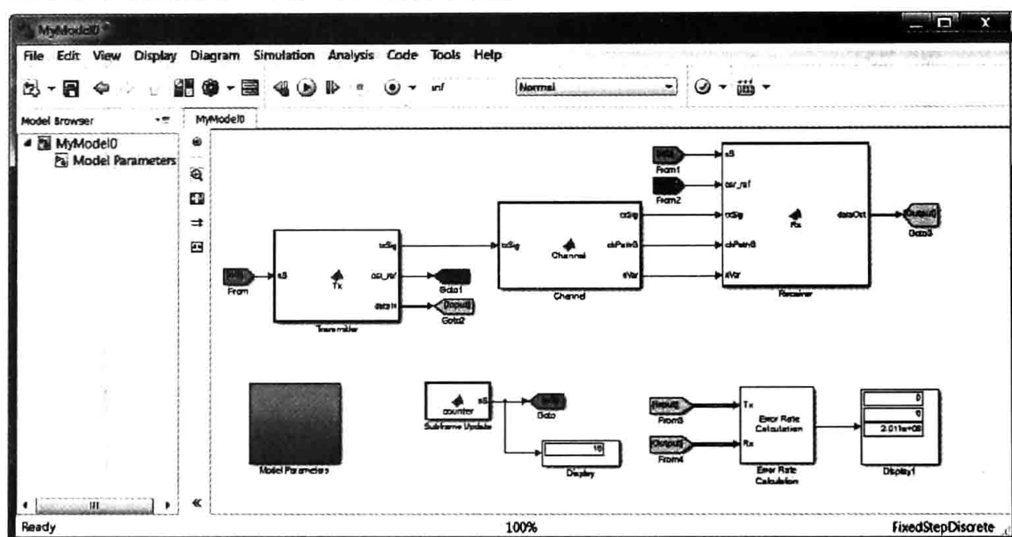


图 8.33　改变子系统背景色并重命名为 Model Parameters

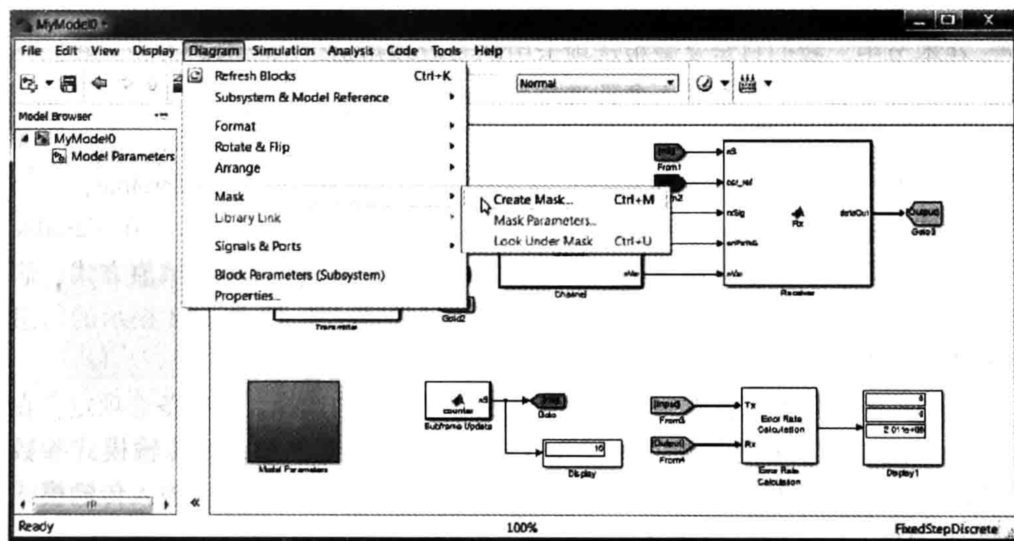


图 8.34　将 Model Parameters 子系统转换为蒙板

蒙板编辑器包含四个选项卡：Icon and Ports 选项卡允许我们在子系统图标上显示文字和图像，Parameters 选项卡允许我们添加参数并定义取值范围，Initialization 选项卡允许我们输入 MATLAB 函数处理子系统参数并帮助我们在 MATLAB

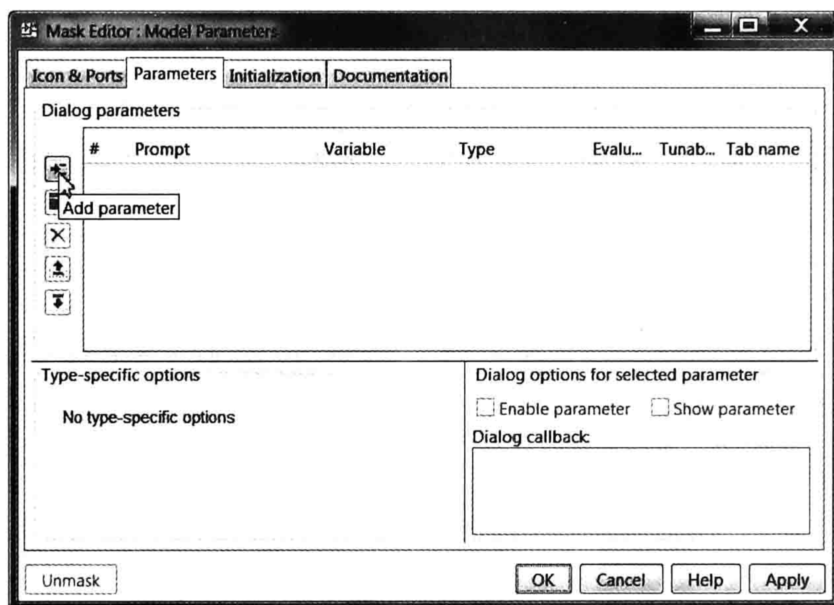


图 8.35　打开 Model Parameters 蒙板编辑器添加参数

工作区中创建各种系统参数，Documentation 选项卡允许当双击子系统图标打开对话框时显示子系统相关的文字信息。

在本节中，我们自定义蒙板选项卡中的参数和初始化选项卡。首先，我们将我们的系统参数一个一个导入参数选项卡的对话框参数列表中，如图 8.36 所示。我们点击对话框参数列表左上的第一个图标，即 Add parameter 添加一个新的参数。对话框参数列表中这时出现一个新的表头，包括如 Prompt、Variable，Type 等等分别标示相应区域。在 Prompt 区，我们输入参数的文字提示。在 Variable 区，我们输入用于初始化阶段的 MATLAB 变量名。Type 区为参数赋值方式。假如我们选择复选框方式，则这个参数会定义为布尔类型，即 Prompt 提示的信息真或假。

假如我们选择弹窗方式，则参数只能从有限个值中选择。这些备选项定义在 Parameter 选项卡左下角 Type - Specific Options 列表中。例如，对传输模式参数（txMode），如图 8.36 所示，我们选择一个弹窗并在备选项列表中输入传输模式（SIMO、TD、开环 SM、闭环 SM）。

现在我们可以观察参数对话框设定第一个参数的结果。我们保存并关闭蒙板编辑器，返回模型界面，双击 Model Parameter 子系统图标。如图 8.37 所示，屏幕出现一个叫做 Model Parameter 的组件参数对话框。里面有一个参数项（Transmission Mode），对应参数内容提示，备选项列表中有我们刚才输入的备选项类型。

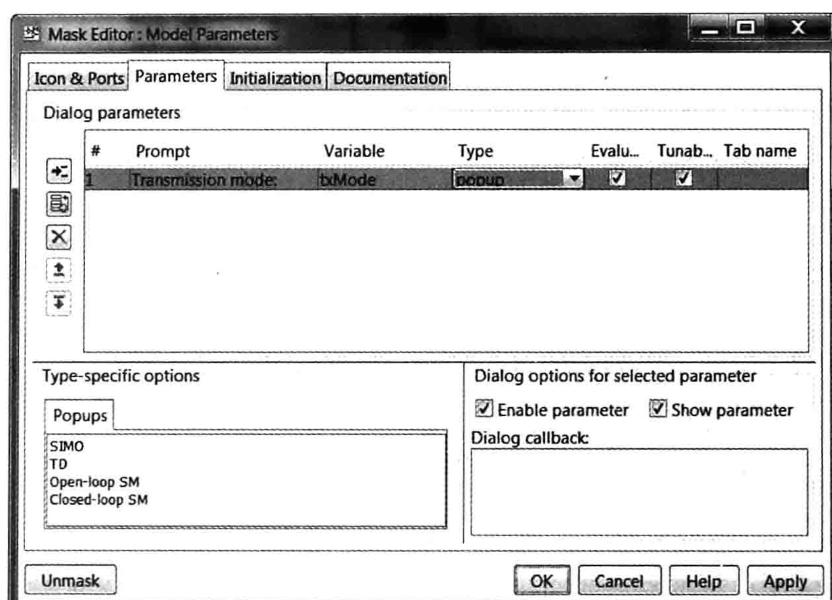


图 8.36　向 Parameter 选项卡对话框参数列表中添加参数

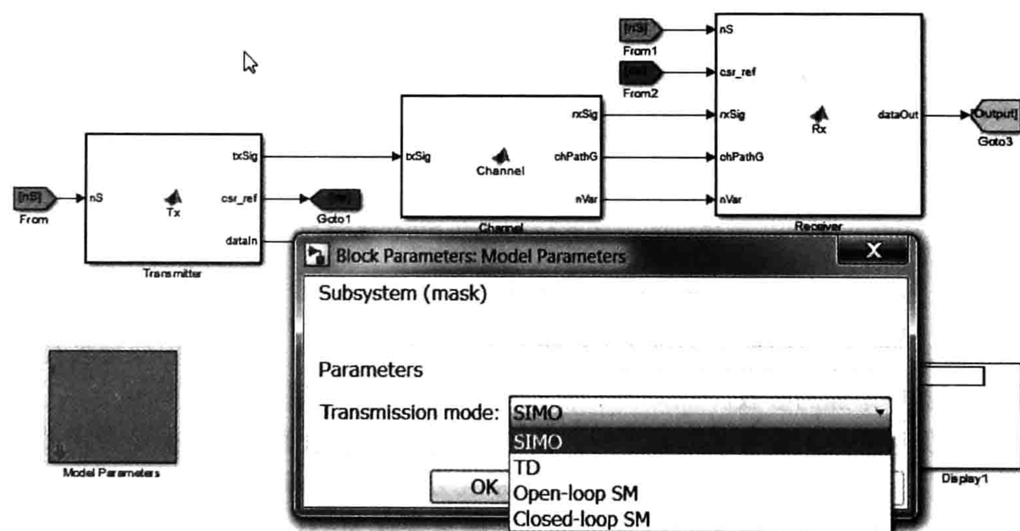


图 8.37　检查参数对话框是否反映了蒙板编辑器添加的参数

我们现在可以重复向蒙板编辑器的参数列表添加参数。注意需要添加 MATLAB 脚本（commlteSystem _ params）中众多参数以生成三个 LTE 参数结构体（prmLTEDLSCH、prmLTEPDSCH、prmMdl）和三个附加参数（snrdB、maxNumBits，maxNumErrs），以供 Simulink 进行系统仿真。下面为 MATLAB 参数脚本（commlteSystem _ params）以供参考：

Algorithm

commlteMIMO_Simulink_init function

```
%% Set simulation parametrs & initialize parameter structures
txMode          = 4;  % Transmisson mode one of {1, 2, 3, 4}
numTx           = 2;  % Number of transmit antennas
numRx           = 2;  % Number of receive antennas
chanBW          = 4;  % [1,2,3,4,5,6] maps to [1.4, 3, 5, 10, 15, 20]MHz
contReg         = 2;  % {1,2,3} for >=10MHz, {2,3,4} for <10Mhz
modType         = 2;  % [1,2,3] maps to ['QPSK','16QAM','64QAM']
numCodeWords = 1;  % Number of codewords in PDSCH
% DLSCH
cRate           = 1/2;  % Rate matching target coding rate
maxIter         = 6;  % Maximum number of turbo decoding terations
fullDecode      = 0;  % Whether "full" or "early stopping" turbo decoding is performed
% Channel
chanMdl         = 'EPA 0Hz';
% one of {'flat','frequency-selective', 'EPA 0Hz', 'EPA 5Hz', 'EVA 5Hz', 'EVA 70Hz'}
Doppler      = 0;                  % a value between 0 to 300 = Maximum Doppler shift
corrLvl         = 'Low';
%  one of {'Low', 'Medium', 'High'} Spatial correlation level between antennas
enPMIfback   = 0;  % Enable/Disable Precoder Matrix Indicator (PMI) feedback
cbIdx           = 1;  % Initialize PMI index
% Simulation parametrs
Eqmode          = 2;  % Type of equalizer used [1,2,3] for ['ZF', 'MMSE','Sphere Decoder']
chEstOn         = 1;     % use channel estimation or ideal channel
snrdB = 12.1;
maxNumErrs      = 2e6; % Maximum number of errors found before simulation stops
maxNumBits      = 2e6; % Maximum number of bits processed before simulation stops
```

图 8.38 表示了按照 MATLAB 参数脚本 commlteSystem _ param 中变量名在蒙板编辑器中添加的参数。

在保存并关闭蒙板编辑器之后，我们可以再一次检查参数对话框是否定义了所有参数。如图 8.39 所示，通过蒙板子系统方式在 Simulink 中定义参数方便易用。我们不再需要先在 MATLAB 编辑器中编辑保存 MATLAB 参数脚本，再返回 Simulink 模型重启仿真这样麻烦的步骤。所有参数现在都可再 Simulink 模型中使用参数对话框定义。

我们很快会发现，这种匹配 MATLAB 参数脚本变量名在蒙板编辑器添加参数的方式，对参数初始化处理也有好处。现在只需要运行 Initialization 选项卡中的初始化命令即可根据参数对话框设定生成 LTE 参数结构体。如图 8.40 所示，Initialization 选项卡左侧列出了参数对话框变量。在右侧我们可以看到 Initialization Commands 编辑窗口。在这个编辑窗口中，我们可以键入各种 MATLAB 命令

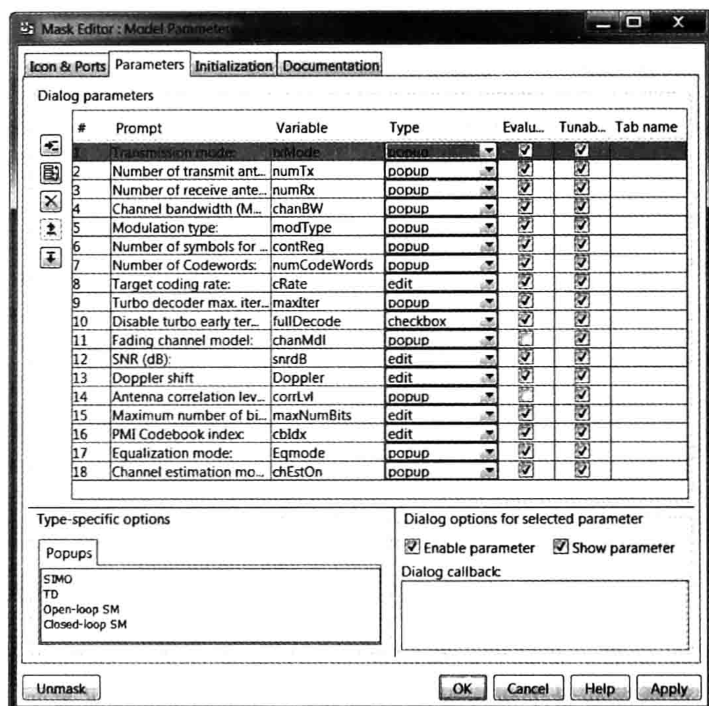


图 8.38 按照 MATLAB 脚本中变量名在参数列表中添加参数

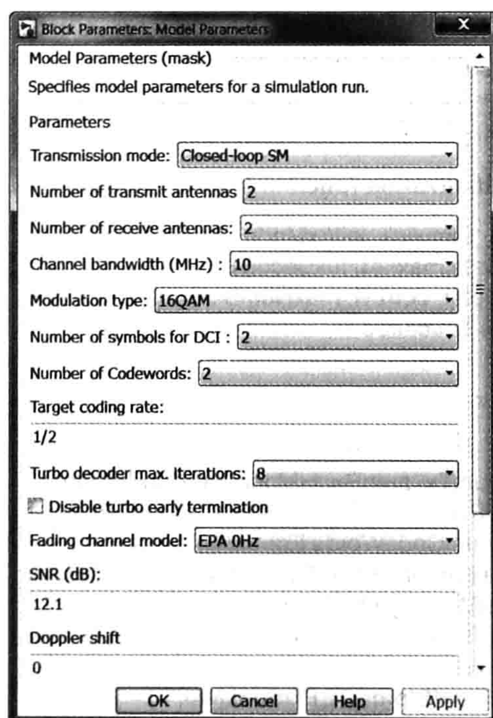


图 8.39 在 Simulink 中设定 LTE 系统模型参数的参数对话框

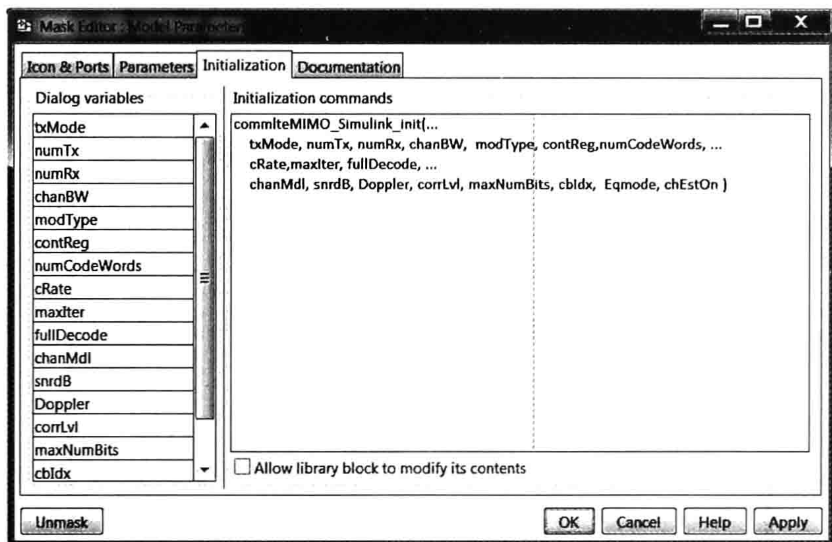


图 8.40 蒙板编辑器中的初始化函数，用于从参数对话框生成 LTE 参数结构体

或调用 MATLAB 函数。在这里，我们调用 MATLAB 函数（commlteMIMO _ Simulink _ init）从对话框变量生成模型参数，并在 MATLAB 工作区显示。

因我们在蒙板编辑器中添加的参数名与 MATLAB 参数脚本相同，故蒙板子系统初始化函数 commlteMIMO _ Simulink _ init 与前文中的 MATLAB 系统模型初始化函数（commlteSystem _ initialize）基本相同。

Algorithm

Masked subsystem initialization function (*commlteMIMO_Simulink_init*)

```
function commlteMIMO_Simulink_init(txMode, Tx, Rx, chanBW, modType, contReg,...
    numCodeWords, cRate,maxIter, fullDecode, ...
   chanMdl, snrdB, Doppler, corrLvl, maxNumBits, cbIdx,  Eqmode, chEstOn )
% Create the parameter structures
vector=[1,2,4];
numTx=vector(Tx);
numRx=vector(Rx);
% PDSCH parameters
CheckAntennaConfig(numTx, numRx, txMode, numCodeWords);
prmLTEPDSCH = prmsPDSCH(txMode, chanBW, contReg, mod-
Type, numTx, numRx, numCodeWords,Eqmode);
[SymbolMap, Constellation]=ModulatorDetail(prmLTEPDSCH.modType);
prmLTEPDSCH.SymbolMap=SymbolMap;
prmLTEPDSCH.Constellation=Constellation;
if numTx==1
   prmLTEPDSCH.csrSize=[2*prmLTEPDSCH.Nrb, 4];
else
```

```
    prmLTEPDSCH.csrSize=[2*prmLTEPDSCH.Nrb, 4, numTx];
end
% DLSCH parameters
prmLTEDLSCH = prmsDLSCH(cRate,maxIter, fullDecode, prmLTEPDSCH);
% Channel parameters
chanSRate   = prmLTEPDSCH.chanSRate;
DelaySpread = prmLTEPDSCH.cpLenR;
 prmMdl = prmsMdl( chanSRate,  DelaySpread, chanMdl, Doppler, numTx, numRx, ...
    corrLvl, chEstOn-1, 0, cbIdx);
%% Assign parameter structure variables to base workspace
assignin('base', 'prmLTEPDSCH', prmLTEPDSCH);
assignin('base', 'prmLTEDLSCH', prmLTEDLSCH);
assignin('base', 'prmMdl', prmMdl);
assignin('base', 'snrdB', snrdB);
assignin('base', 'maxNumBits', maxNumBits);
assignin('base', 'maxNumErrs', maxNumBits);
```

注意这个初始化函数与我们前文中使用的 MATLAB 模型的不同之处在于最后多出来的六行代码。这些额外代码定义变量为本地类型并将它们输出到 MATLAB 工作区。

8.6 定量评估

在本章最后一节中，我们对 LTE 系统模型进行定量评估。与处理随机生成的载荷比特不同，我们将处理声音信号的比特流。在一定意义上，这个仿真模拟了 LTE PHY 模型实现手机通话的情形。

8.6.1 声音信号传输

进行定量评估的第一步是引入声音编码。在这一步中，我们将声音信号编码，并将编码比特流作为 LTE 收发端模型的输入。我们采用一种最简单的声音编码算法——基于 A 律和 μ 律码的调制（PCM）编码。在接收端，我们用基于 A 律和 μ 律码译码器复原 LTE 模型处理的比特流，以得到输出声音信号并收听它。复原的声音信号质量反映了信道和接收端带来的所有劣化影响。

为了模拟 LTE 手机通话，我们用上一节构建的 Simulink 模型。唯一一点需要更新的地方（见图 8.41）即音声信号编码和译码：

1）从发射端子系统中移除载荷比特生成器函数；

2）引入编码声音信号作为发射端组件输入；

3）声音编码：生成声音编码比特，一个一个子帧通过 DSP 系统工具箱组件；

4）声音译码：对 LTE 系统模型输出进行译码并复原出输出声音信号。

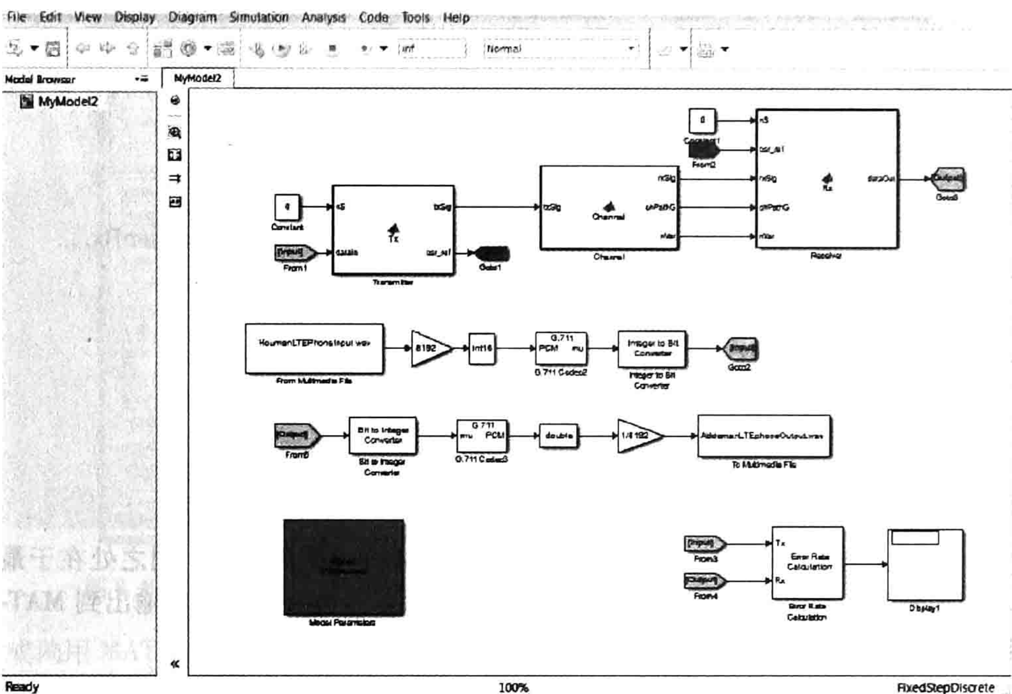


图 8.41　在 LTE 收发端 Simulink 模型中添加音声编码和解码组件以测量音声质量

声音编码操作按如下顺序进行：

1）用 DSP 系统工具箱的 From Multimedia File Block 组件从音频文件（任何 MATLAB 支持的音频格式文件）中抽取原始声音子帧流。

2）使用 μ 率编码器（值为 8192）处理 From Multimedia File Block 组件归一化的音频采样输出，并将结果保存为整型（int16）MATLAB 数据类型。

3）将整型输入通过 DSP 系统工具箱的 G. 711 PCM 编码器组件进行处理。

4）用通信系统工具箱中的整型 - 比特转换组件，将压缩数据字节解压到一个个比特中。该组件输出的编码比特流随后作为发射端组件的输入读入 LTE 收发端模型的输入端口。

声音译码操作顺序与编码顺序相反：

1）用比特 - 整型转换器组件将 LTE 收发器模型输出比特打包为字节；

2）将打包字节通过 G. 711 PCM 组件；

3）将 G. 711 PCM 采样转换为浮点数据类型并将其值归一化为 1 和 - 1；

4）用 From Multimedia File Block 组件将输出采样写入磁盘中的音频文件。

8. 6. 2　主观声音质量测试

我们可以在各种条件下进行模型仿真，包括 SIMO、发射分集，和空分复用

传输模式。当我们听到音声文件输出时，注意音声信号的质量与收发器和信道模型参数的相关性。比如，考虑衰落迟延扩散情况——反映在衰落信道路径迟延参数——为 4.6μs（标准定义），并设定复原音声的 SNR 为一个典型值，则我们可以听到声音很清晰。与此类似，当采用提升链路质量的模式时，如使用发射分集替代 SISO（单输入单输出）时，我们可以听到更清晰的声音。

8.7 本章小结

在本章中，我们构建了 LTE PHY 模型的系统模型。我们将前四种下行链路传输模式集成系统模型，包括发射端、信道模型，和接收端。然后我们对系统模型进行仿真以定量评估整体系统性能。我们研究了各个传输模式、信道模型、链路 SNR、信道估计技术、MIMO 接收器算法，和信道迟延扩散对整体性能的影响。我们也通过仿真衡量了 LTE 系统模型的吞吐量。

随后，我们为 LTE 收发端模型构建了 Simulink 模型。我们一步步创建 Simulink 模型，将发射端、接收端，和信道模型的 MATLAB 函数一步步集成进 Simulink 系统模型中。我们随后通过开发参数对话框方便的定义了 LTE 系统模型参数。最后，我们在 Simulink 模型中添加了音声编码器和译码器以定量分析系统性能。

参考文献

[1] Jafarkhani, H. (2005) *Space-time Coding; Theory and Practice*, Cambridge University Press, New York.
[2] 3GPP Evolved Universal Terrestrial Radio Access (E-UTRA); User Equipment (UE) Radio Transmission and Reception (Version 11.4.0), March 2013. TS 36.101.

9 仿　　真

在前面各章中我们对 LTE 的 PHY 层标准进行功能描述并用 MATLAB 实现。为了验证这个模型能否满足标准化处理的要求，我们需要进行大规模仿真。如其他很多标准，LTE 标准为模式化协议。这意味着我们需要对所有可能的模式组合进行仿真进行确认，包括调制、编码和 MIMO 各模式的组合。这些组合所带来的大数据量仿真设定和计算复杂度会不可避免地带来如下挑战：超长的仿真时间以及加速仿真速度的必要性。

仿真可以在软件模型或物理硬件原型上进行。大多数设计者都希望在硬件原型验证之前，利用标准的计算机模型对系统性能相关的技术层面进行验证。当我们设计如何加速软件模型执行速度时，我们会自然而然地设定一个基准或初期版本。对这个基准算法仿真的加速优化工作可能会影响模型的功能精度，也可能不造成任何影响。为了真实的按标准实现，在本书中我们重点在保证基准算法的数值精度前提下进行优化。因此，优化工作将重点通过各种方法保证性能的同时提高效率。在本章中，我们详细讨论各种 MATLAB 和 Simulink 优化方法，以大幅度提升仿真速度。

9.1 提升 MATLAB 仿真速度

当我们对一个通信系统建模时，我们的关注点会根据不同阶段而各不一样。在开发早期，我们关注数学模型是否精确。在这一阶段我们要使用 MATLAB 可视化和调试功能，验证 MATLAB 函数和脚本描述的操作处理是否正确。在这一阶段的验证有时为单元测试和已知目标结果的有限数据测试。单元测试帮助确认数学模型是否真实反映了设计。在单元测试通过之后，大部分设计者会在仿真循环中设定大数据量条件，代入相同模型进行仿真。在大规模仿真中明确设计瓶颈能使优化得到更大的效果。我们可以优化基准模型，然后通过下面的一种方法解决设计瓶颈（见图 9.1）。

MATLAB 代码优化：包括更新 MATLAB 程序代码使其更有效的执行。这个工作包含两个步骤：

1）确保常数参数只在初始化时调用计算；

2）减少参数校验开销；

3）用预分配地址的变量代替动态内存分配以减小开销；

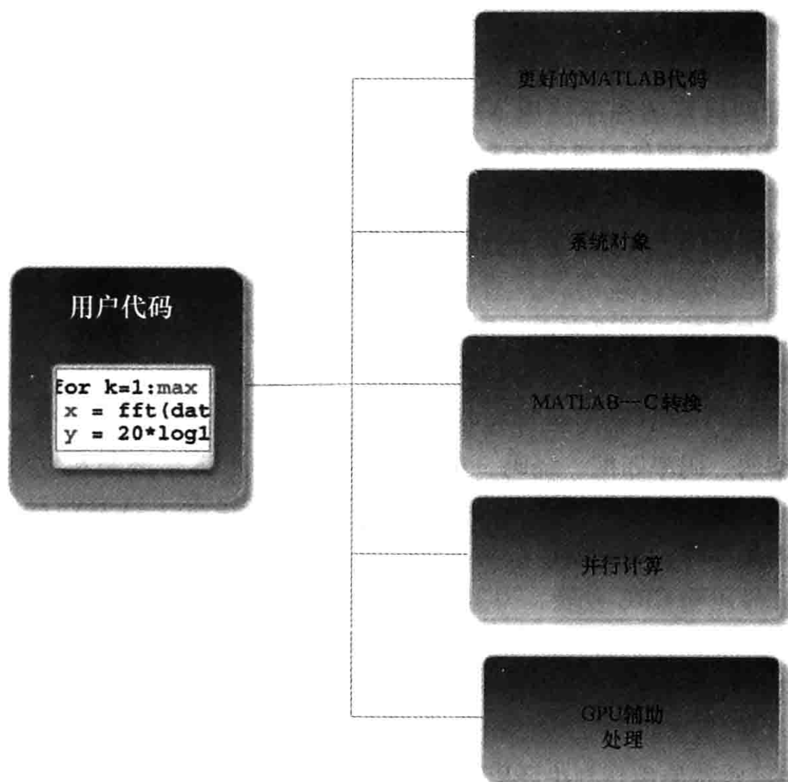


图 9.1　MATLAB 仿真加速方法

4）使用系统对象中更高执行效率的算法。

包括如下技术：

1）将 MATLAB 代码转换编译为 C 代码；

2）使用多核或并行处理束；

3）使用图像处理单元（GPU）处理 MATLAB 程序。

9.2　工作流程

在本章中，我们针对基准 MATLAB 程序进行一系列的代码优化，从而提高仿真速度。在每一步中，每个算法生成的数值输出都相同。唯一不同的是随着每一步优化，程序使用了更高效的编程技术。

本书中得出的数值和时间结果根据 MATLAB 运行平台、操作系统类型、C/C + + 编译器或 GPU 使用与否而不同。本书中无 GPU 辅助运算的 MATLAB 程序在一个无笔记本电脑上运行，其配置环境如下：

硬件：2.70GHz Intel Dual – Core i7 – 2620M CPU，8GB RAM

操作系统：64 位 Windows 7 Enterprise（SP1）

C/C + + 编译器：Microsoft Visual Studio 2010 与 Microsoft Windows SDK7. 1

GPU 辅助运算的程序使用 NVIDIA Tesla GPU 加速器、Intel Quad - core i7 CPU、12GB RAM 硬件配置的台式计算机。其操作系统和编译器与前面相同。

9. 3　实例研究：LTE PDCCH 处理

我们在本章的实例研究中，用一个简化的信号处理模型表征 LTE 标准的下行链路控制信道（PDCCH）。我们已经在第 7 章中讲解过这个算法。如图 9. 2 所示，在发射端 PDCCH 信号处理包括如下步骤：生成循环冗余校验（CRC）、尾比特卷积编码、码率匹配、绕码、正交相移键控（QPSK）调制和发射分集 MIMO 编码。信道建模包括 2 ×2 MIMO 信道以及白高斯噪声（AWGN）信道。接收端反转发射端操作，包括发射分集 MIMO 合并、QPSK 解调、解绕码、码率解匹配、Vitetbi 译码和 CRC 校验。为了减小算法的复杂度，在本节中我们会进行两点处理：

1）忽略频域传输包括正交频分复用（OFDM）资源网格构建和信号生成步骤；

2）在接收端使用硬判决译码。

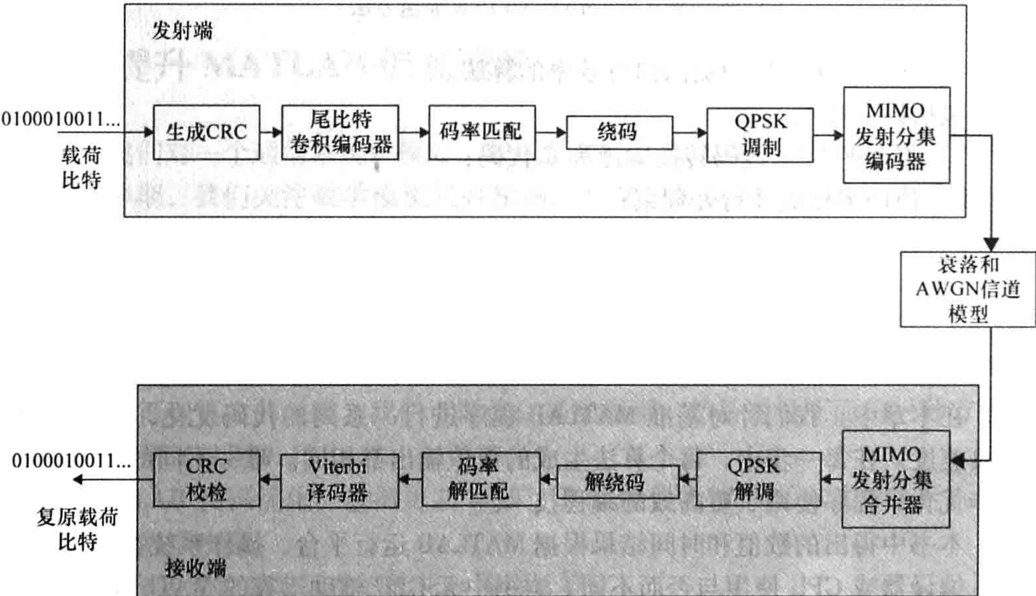


图 9. 2　一个简化的 PDCCH 信号处理链

9.4 基准算法

下面的基准函数为实现 PDCCH 处理链的原始版本。其描述的 PDCCH 算法已在前面各章中详述。一些函数（如 convenc 和 vitdec）以及对象（如 modem. pskmod 和 crc. generator）可以在通信系统工具箱中找到。另外，如 TransmitDiversityEncoder1 和 MIMOFadingChan，由用户自定义和 MATLAB 基础函数以及结构构成。

Algorithm

MATLAB function

```
function [ber, bits]=zPDCCH_v1(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2048;
M=4; k=log2(M); codeRate=1/3;
snr = EbNo + 10*log10(k) + 10*log10(codeRate);
trellis=poly2trellis(7, [133 171 165]);
L=FRM+24;C=6; Index=[L+1:(3*L/2) (L/2+1):L];
%% Initializations
persistent Modulator Demodulator CRCgen CRCdet
if isempty(Modulator)
   Modulator=modem.pskmod('M', 4, 'PhaseOffset', pi/4, 'SymbolOrder', 'Gray',
'InputType', 'Bit');
   Demodulator= modem.pskdemod('M', 4, 'PhaseOffset', pi/4, 'SymbolOrder', 'Gray',
'OutputType', 'Bit');
   CRCgen = crc.generator([1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
   CRCdet = crc.detector  ([1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
end
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u          = randi([0 1], FRM,1);
   u1         = generate(CRCgen,u);
   u2         = u1((end-C+1):end);
   [~, state] = convenc(u2,trellis);
   u3         = convenc(u1,trellis,state);
   u31        = fcn_RateMatcher(u3, L, codeRate);
   u32        = fcn_Scrambler(u31, nS);
   u4         = modulate(Modulator, u32);
   u5         = TransmitDiversityEncoder1(u4);
  % Channel model
   [u6, h6]   = MIMOFadingChan(u5);
   u7         = awgn(u6,snr);
```

```
  % Receiver
   u8          = TransmitDiversityCombiner1(u7, h6);
   u9          = demodulate(Demodulator,u8);
   u91         = fcn_Descrambler(u9, nS);
   u92         = fcn_RateDematcher(u91, L);
   uA          = [u999;u999];
   uB          = vitdec(uA ,trellis,34,'trunc','hard');
   uC          = uB(Index);
   y           = detect(CRCdet, uC );
   numErrs     = numErrs + sum(y~=u);
   numBits     = numBits + FRM;
   nS          = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;
bits=numBits;
```

让我们对基准算法进行性能评估。下面的 MATLAB 脚本（zPDCCH _ v1 _ test）在一个 for 循环中执行这个程序。在每一次循环中，脚本调用基准算法赋给定的信噪比（SNR）值并计算相应的比特误码率（BER）。脚本同时用 tic 和 toc 函数测量循环所需时间。

Algorithm

MATLAB script: zPDCCH_v1_test

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\nVersion 1: Baseline algorithm\n\n');
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v1 (snr, MaxNumBits, MaxNumBits);
end
time_1=toc;
fprintf(1,'Version 1: Time to complete %d iterations = %6.4f (sec)\n',
MaxSNR, time_1);
```

当我们运行这个 MATLAB 脚本时，命令界面会提示当前算法版本，当前循环次数和最终总运行时间。其结果如图 9.3 所示。在这个实例中，八次循环处理 1 百万比特数据共需 411.30s。

我们以这个算法的执行结果作为衡量后面代码优化后性能优劣的标尺。在进行代码优化之前，明确代码瓶颈是非常关键的。这一算法上的问题很大程度上影响了计算复杂度和处理时间。我们将使用一些 MATLAB 工具确定我们算法的瓶颈所在。

```
Version 1: Baseline algorithm

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 1: Time to complete 8 iterations = 411.3030 (sec)
```

图 9.3 基准算法：循环八次耗时

9.5 MATLAB 代码剖析

MATLAB 提供了一系列工具评估和优化代码性能。MATLAB 分析器可以找到哪些代码运行耗时长。我们可以用下面三条命令调用这个工具分析基准算法：

Algorithm

MATLAB script

```
profile on;
ber= zPDCCH_v1 (snr, MaxNumBits, MaxNumBits);
profile viewer;
```

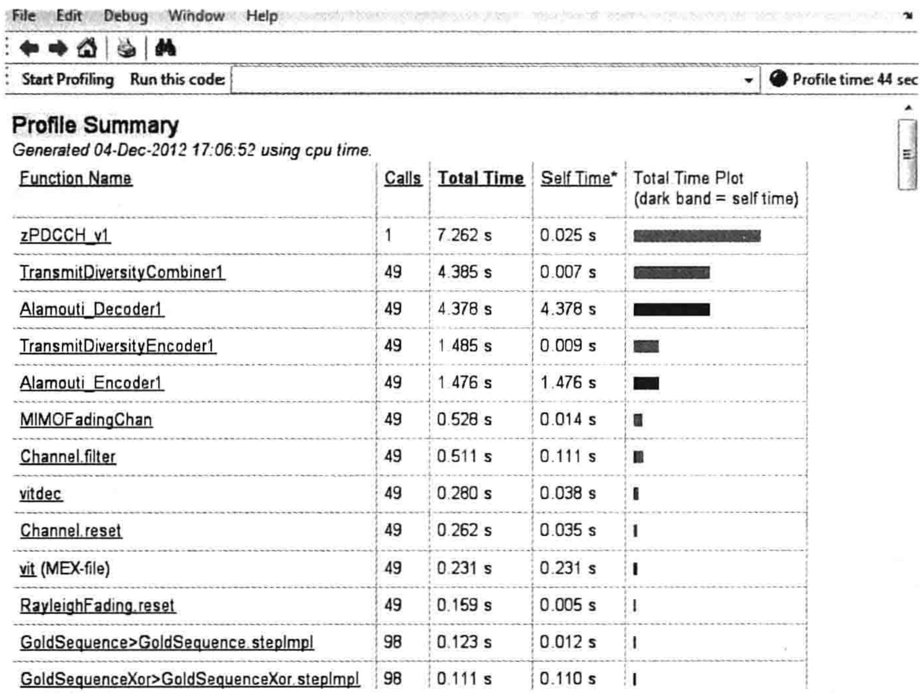
通过调用 profile viewer 命令启动 MATLAB 分析器得到分析报告，如图 9.4 所示。MATLAB 分析器对代码总体运行状况给出一个统计报告，包括所有的函数调用列表、每个函数调用的次数，以及每个函数运行的总时间。它还包括每个函数的测时信息，如哪一行代码耗费时间最长。

当瓶颈被确认，我们就可以重点针对这个部分改进性能。例如，分析报告指出，函数 TransmitDiversity - Combiner 耗费 4.385s，而全体运行时间为 7.262s。因此，确定 TransmitDiversityCombiner 为基准算法的一个瓶颈。

Algorithm

MATLAB function

```
function y = TransmitDiversityCombiner1(in, chEst)
%#codegen
% Alamouti Transmit Diversity Combiner
% Scale
in = sqrt(2) * in;
% STBC Alamouti
y =  Alamouti_Decoder1(in, chEst);
% Space-Frequency to Space-Time transformation
y(2:2:end) = -conj(y(2:2:end));
```


File Edit Debug Window Help

Start Profiling Run this code: Profile time: 44 sec

Profile Summary

Generated 04-Dec-2012 17:06:52 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
zPDCCH_v1	1	7.262 s	0.025 s	
TransmitDiversityCombiner1	49	4.385 s	0.007 s	
Alamouti_Decoder1	49	4.378 s	4.378 s	
TransmitDiversityEncoder1	49	1.485 s	0.009 s	
Alamouti_Encoder1	49	1.476 s	1.476 s	
MIMOFadingChan	49	0.528 s	0.014 s	
Channel.filter	49	0.511 s	0.111 s	
vitdec	49	0.280 s	0.038 s	
Channel.reset	49	0.262 s	0.035 s	
vit (MEX-file)	49	0.231 s	0.231 s	
RayleighFading.reset	49	0.159 s	0.005 s	
GoldSequence>GoldSequence.stepImpl	98	0.123 s	0.012 s	
GoldSequenceXor>GoldSequenceXor.stepImpl	98	0.111 s	0.110 s	

图 9.4　基准算法的分析总结报告

当我们在分析总结报告中点击函数超链接时，我们可以找到 TransmitDiversityCombiner1 中那一行代码耗时最长。可以轻松发现，在三行代码中 Alamouti_Decoder1 为瓶颈。下面的 Alamouti_Decoder1 函数即实现 Alamouti 合并算法的第一个版本。

Algorithm

MATLAB function

```
function s = Alamouti_Decoder1(u,H)
%#codegen
% STBC_DEC STBC Combiner
%   Outputs the recovered symbol vector
LEN=size(u,1);
Nr=size(u,2);
BlkSize=2;
NoBlks=LEN/BlkSize;
% Initialize outputs
h=complex(zeros(1,2));
s=complex(zeros(LEN,1));
% Alamouti code for 2 Tx
indexU=(1:BlkSize);
for m=1:NoBlks
```

```
    t_hat=complex(zeros(BlkSize,1));
    h_norm=0.0;
    for n=1:Nr
      h(:)=H(2*m-1,:,n);
      h_norm=h_norm+real(h*h');
      r=u(indexU,n);
      r(2)=conj(r(2));
      shat=[conj(h(1)), h(2); conj(h(2)), -h(1)]*r;
      t_hat=t_hat+shat;
    end
    s(indexU)=t_hat/h_norm; % Maximum-likelihood combining
    indexU=indexU+BlkSize;
  end
end
```

通过 Alamouti _ Decoder1 函数的超链接，我们可以更详细的逐行分析其耗时（见图 9.5）。这种逐级分析可以使我们明确哪一部分代码对性能影响最大。在本实例中。算法内的两个 For 循环嵌套和逐个计算向量元素，以及标量化部分耗时最长。向量化处理这部分代码可以提升速度。

```
 time     calls  line
                   1 function s = Alamouti_Decoder1(u,H)
                   2 %#codegen
                   3 % STBC_DEC STBC Combiner
                   4 %   Outputs the recovered symbol vector
              49   5 LEN=size(u,1);
              49   6 Nr=size(u,2);
              49   7 BlkSize=2;
              49   8 NoBlks=LEN/BlkSize;
                   9 % Initialize outputs
              49  10 h=complex(zeros(1,2));
< 0.01        49  11 s=complex(zeros(LEN,1));
                  12 % Alamouti code for 2 Tx
              49  13 indexU=(1:BlkSize);
              49  14 for m=1:NoBlks
  0.24     76146  15     t_hat=complex(zeros(BlkSize,1));
  0.08     76146  16     h_norm=0.0;
  0.05     76146  17     for n=1:Nr
  0.52    152292  18         h(:)=H(2*m-1,:,n);
  0.55    152292  19         h_norm=h_norm+real(h*h');
  0.31    152292  20         r=u(indexU,n);
  0.07    152292  21         r(2)=conj(r(2));
  1.65    152292  22
  0.21    152292  23         t_hat=t_hat+shat;
  0.03    152292  24     end
  0.60     76146  25     s(indexU)=t_hat/h_norm; % Maximum-likelihood combining
  0.05     76146  26     indexU=indexU+BlkSize;
  0.01     76146  27 end
              49  28 end
```

图 9.5 Alamouti _ Decoder1 函数每一行代码处理耗时

9.6 MATLAB 代码优化

在本节中我们讨论一些常用的 MATLAB 代码优化技术。这些技术包括向量化处理代码、预分配数据、分离循环内的初始化处理，和使用系统对象。我们将在优化 PDCCH 处理算法的过程中逐一展示这些技术。

9.6.1 向量化

向量化是 MATLAB 中代码优化的重要技术。通过向量化过程，我们将循环

体转变为矩阵和向量操作。因 MATLAB 为矩阵和向量计算配置了处理器优化库，所以我们经常可以通过向量化代码提升性能。

PDCCH 算法现在通过向量化进行优化。这个优化的版本和基准的唯一区别是 TransmitDiversityCombiner2 替代了 TransmitDiversityCombiner1。这个新函数是发射分集合并函数的新版本，包含了一个 Alamouti _ Decoder2 函数，为 Alamouti _ Decoder1 函数向量化的结果。我们考察 Alamouti _ Decoder2 函数，可以发现两个嵌套的 For 循环变为一个 For 循环，其循环内处理更加向量化。对比如下：

MATLAB function

function [ber, bits]=zPDCCH_v1(...) . .	function [ber, bits]=zPDCCH_v2(...) . .
u5 = TransmitDiversityDecoder1(u4);	u5 = TransmitDiversityDecoder2(u4);
. . end	. . end
function y = TransmitDiversityCombiner1(in, chEst) %#codegen % Alamouti Transmit Diversity Combiner % Scale in = sqrt(2) * in; % STBC Alamouti	function y = TransmitDiversityCombiner2(in, chEst) %#codegen % Alamouti Transmit Diversity Combiner % Scale in = sqrt(2) * in; % STBC Alamouti
y = Alamouti_Decoder1(in, chEst);	y = Alamouti_Decoder2(in, chEst);
% Space-Frequency to Space-Time transformation y(2:2:end) = -conj(y(2:2:end));	% Space-Frequency to Space-Time transformation y(2:2:end) = -conj(y(2:2:end));

function s = Alamouti_Decoder1(u,H)
LEN=size(u,1);
Nr=size(u,2);
BlkSize=2;
NoBlks=LEN/BlkSize;
% Initialize outputs
h=complex(zeros(1,2));
s=complex(zeros(LEN,1));
% Alamouti code for 2 Tx
indexU=(1:BlkSize);
	function s = Alamouti_Decoder2(u,H)
	LEN=size(u,1);
	BlkSize=2;
	NoBlks=LEN/BlkSize;
	T=[0 1;-1 0];
	% Initialize outputs
	s=complex(zeros(LEN,1));
	% Alamouti code for 2 Tx
	h=complex(zeros(BlkSize,BlkSize));

for m=1:NoBlks
  t_hat=complex(zeros(BlkSize,1));
  h_norm=0.0;
  for n=1:Nr
	for m=1:NoBlks

h(:)=H(2*m-1,:,n);
    h_norm=h_norm+real(h*h');
    r=u(indexU,n);
    r(2)=conj(r(2));
	indexU=(m-1)*BlkSize+(1:BlkSize);
	  h(:)=H(2*m-1,:,:);
	  h_norm=sum(h(:).*conj(h(:)));
	  r=u(indexU,:);
	  r(2,:)=conj(r(2,:));
	  H1=conj(h);
	  H2=T*h;
	  M=[H1(:,1),H2(:,1),H1(:,2),H2(:,2)];

```
    shat=[conj(h(1)), h(2); conj(h(2)), -h(1)]*r;
    t_hat=t_hat+shat;
  end
  s(indexU)=t_hat/h_norm; % Maximum-likelihood combining
  indexU=indexU+BlkSize;
end
end
```

```
    s(indexU)=(M*r(:))/h_norm; % Maximum-likelihood combining
  end
```

为了验证这些优化是否对耗时有改进，我们运行如下 MATLAB 脚本。脚本除了调用的 PDCCH 算法替换为优化后的版本（zPDCCH _ v2. m）之外，与前面的脚本相同。其结果显示，八次循环处理 1 百万比特数据共需 326. 50s（见图 9. 6）。

```
Version 2: Vectorization

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 2: Time to complete 8 iterations = 326.5071 (sec)
```

图 9. 6 算法的第二个版本：循环八次耗时

Algorithm

MATLAB script: zPDCCH_v2_test

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\nVersion 1: Baseline algorithm\n\n');
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v2 (snr, MaxNumBits, MaxNumBits);
end
time_2=toc;
fprintf(1,'Version 1: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_2);
```

算法的第二个版本使用一个 for 循环和一个表征循环次数的变量 NoBlks，它和函数的第一阶输入有关。函数第一阶输入非常庞大：在本算法中为 3108。第一阶输入频繁循环以及对小长度向量进行向量化操作不会对向量化优化带来任何好处。

算法的第三个版本对输入的第一阶进行向量化。在这一版中，我们只循环两次（如第二阶一样）、并对第一阶输入有关的大向量和矩阵进行向量化。基于大向量和矩阵的代码向量化可以得到更好的优化结果。两个算法的区别在于用 TransmitDiversityCombiner3 替换 TransmitDiversityCombiner2。

MATLAB function

```
function [ber, bits]=zPDCCH_v2(...)
.
.
u5 = TransmitDiversityDecoder2(u4);
.
.
end

function y = TransmitDiversityCombiner2(in,
   chEst)
%#codegen
% Alamouti Transmit Diversity Combiner
% Scale
in = sqrt(2) * in;
% STBC Alamouti
y =  Alamouti_Decoder2(in, chEst);
% Space-Frequency to Space-
   Time transformation
y(2:2:end) = -conj(y(2:2:end));

function s = Alamouti_Decoder2(u,H)
LEN=size(u,1);
BlkSize=2;
function s = Alamouti_Decoder2(u,H)
LEN=size(u,1);
BlkSize=2;
NoBlks=LEN/BlkSize;
T=[0 1;-1 0];
% Initialize outputs
s=complex(zeros(LEN,1));
% Alamouti code for 2 Tx
h=complex(zeros(BlkSize,BlkSize));
for m=1:NoBlks
   indexU=(m-1)*BlkSize+(1:BlkSize);
   h(:)=H(2*m-1,:,:);
   h_norm=sum(h(:).*conj(h(:)));
   r=u(indexU,:);
```

```
function [ber, bits]=zPDCCH_v3(...)
.
.
u5 = TransmitDiversityDecoder3(u4);
.
.
end

function y = TransmitDiversityCombiner3(in,
   chEst)
%#codegen
% Alamouti Transmit Diversity Combiner
% Scale
in = sqrt(2) * in;
% STBC Alamouti
y =  Alamouti_Decoder3(in, chEst);
% Space-Frequency to Space-
   Time transformation
y(2:2:end) = -conj(y(2:2:end));

function y = Alamouti_Decoder3(u,Ch)
%#codegen
% STBC_DEC STBC Combiner
LEN=size(u,1);
BlkSize=2;
NoBlks=LEN/BlkSize;
Nr=size(u,2);
idx1=1:BlkSize:LEN;
idx2=idx1+1;
% Initalize outputs
s=complex(zeros(LEN,Nr));
mynorm=complex(zeros(LEN,BlkSize));
vec_u=complex(zeros(NoBlks,BlkSize));
% Alamouti code for 2 Tx
H=complex(zeros(NoBlks,BlkSize));
for n=1:Nr
   vec_u(:,1)        = u(idx1,n);
```

```
    r(2,:)=conj(r(2,:));
    H1=conj(h);
    H2=T*h;
    M=[H1(:,1),H2(:,1),H1(:,2),H2(:,2)];
    s(indexU)=(M*r(:))/h_norm; % Maximum-
    likelihood combining
end
```

```
    vec_u(:,2)       = conj(u(idx2,n));
    H(:)             = Ch(1:BlkSize:end,:,n);
    conjH            = conj(H);
    cn1              = [conjH(:,1), H(:,2)];
    s(idx1,n)        = sum(cn1.*vec_u,2);
    mynorm(idx1,n) = sum(H.*conj(H),2);
    cn2              = [conjH(:,2), -H(:,1)];
    s(idx2,n)        = sum(cn2.*vec_u,2);
  end;
  nn=sum(mynorm,2);
  nn(idx2)=nn(idx1);
  y=sum(s,2)./nn;
  end
```

为了验证这些优化是否对耗时有改进，我们运行如下 MATLAB 脚本（zPDCCH _ v3 _ test）。第三个版本的算法耗时 175. 84s 循环八次处理 1 百万比特数据，如图 9. 7 所示。

```
Version 3: Vectorization along larger dimension

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 3: Time to complete 8 iterations = 175.8478 (sec)
```

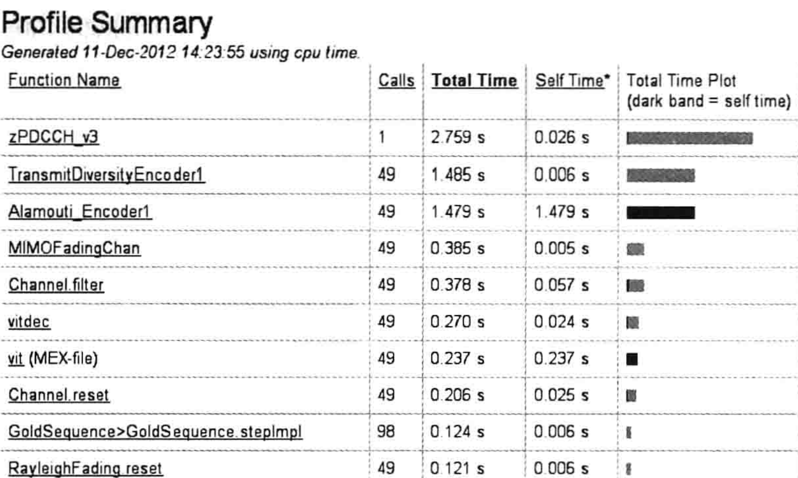
图 9. 7　第三个版本的算法：循环八次耗时

Algorithm

MATLAB script: zPDCCH_v3_test

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\nVersion 3: Better vectorized algorithm\n\n');
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v3 (snr, MaxNumBits, MaxNumBits);
end
time_3=toc;
fprintf(1,'Version 3: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_3);
```

通过分析这个算法的第三个版（zPDCCH _ v3），我们发现 TransmitDiversity-Encoder1 为下一个瓶颈。这个函数调用了 Alamouti 编码器函数的第一版（Alamouti _ Encoder1）。函数分析器命令和结果报告如图 9. 8 所示。



Profile Summary

Generated 11-Dec-2012 14:23:55 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
zPDCCH_v3	1	2.759 s	0.026 s	
TransmitDiversityEncoder1	49	1.485 s	0.006 s	
Alamouti_Encoder1	49	1.479 s	1.479 s	
MIMOFadingChan	49	0.385 s	0.005 s	
Channel.filter	49	0.378 s	0.057 s	
vitdec	49	0.270 s	0.024 s	
vit (MEX-file)	49	0.237 s	0.237 s	
Channel.reset	49	0.206 s	0.025 s	
GoldSequence>GoldSequence.stepImpl	98	0.124 s	0.006 s	
RayleighFading.reset	49	0.121 s	0.006 s	

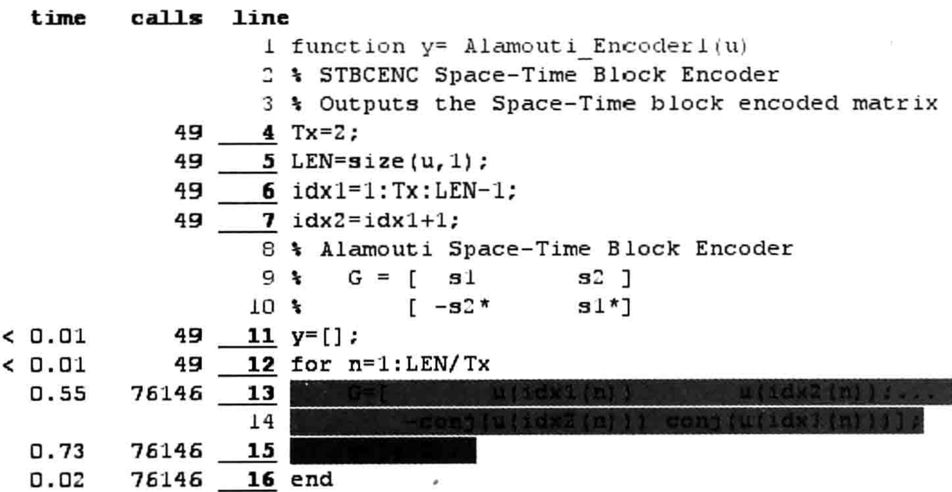
图 9.8　第三个版本算法的分析总结报告

Algorithm

MATLAB script

```
profile on;
ber= zPDCCH_v3(snr, MaxNumBits, MaxNumBits);
profile viewer;
```

通过点击分析报告中 Alamouti _ Encoder1 的超链接，我们可以看到逐行代码的执行耗时分析（见图 9.9）。注意我们初始化输出矩阵 y 为一个空矩阵。在 for 循环中，我们随后将矩阵 y 的阶数扩展为一个 2 ×2 Alamouti 矩阵。在下面的循环中，我们必须分配新内存空间并将原矩阵拷贝到新地址。因此，我们可以知道预分配技术将对代码性能优化起到帮助。下面我们会讨论预分配优化 MATLAB 代码。



```
 time    calls  line
                  1 function y= Alamouti_Encoder1(u)
                  2 % STBCENC Space-Time Block Encoder
                  3 % Outputs the Space-Time block encoded matrix
            49    4 Tx=2;
            49    5 LEN=size(u,1);
            49    6 idx1=1:Tx:LEN-1;
            49    7 idx2=idx1+1;
                  8 % Alamouti Space-Time Block Encoder
                  9 %   G = [  s1        s2 ]
                 10 %         [ -s2*      s1*]
< 0.01      49   11 y=[];
< 0.01      49   12 for n=1:LEN/Tx
  0.55   76146   13     G=[         u(idx1(n))         u(idx2(n));...
                 14           -conj(u(idx2(n)))  conj(u(idx1(n)))];
  0.73   76146   15
  0.02   76146   16 end
```

图 9.9　分析 Alamouti _ Encoder1

9.6.2 预分配

预分配为在程序开始时对已知大小的矩阵进行初始化。它可以防止代码运行过程中矩阵大小被动态更改，特别是在使用 While 循环时。因为矩阵需要相邻的内存区块，重复更改矩阵大小会使 MATLAB 寻找相邻大空间区块并移动矩阵耗费时间。通过预分配矩阵空间，我们可以避免不必要的内存操作从而提速。

PDCCH 算法的第四个版本为基于预分配的优化。这个版本中 TransmitDiversityEncoder2 代替 TransmitDiversityEncoder1；该函数使用发射分集编码器函数的第二个版本，即 Alamouti _ Encoder2，为 Alamouti _ Encoder1 的预分配优化。

Algorithm

MATLAB function

```
function [ber, bits]=zPDCCH_v4(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2048;
M=4; k=log2(M); codeRate=1/3;
snr = EbNo + 10*log10(k) + 10*log10(codeRate);
trellis=poly2trellis(7, [133 171 165]);
L=FRM+24;C=6; Index=[L+1:(3*L/2) (L/2+1):L];
%% Initializations
persistent Modulator Demodulator CRCgen CRCdet
if isempty(Modulator)
   Modulator    = modem.pskmod('M', 4, 'PhaseOffset', pi/4, 'SymbolOrder', 'Gray',
'InputType', 'Bit');
   Demodulator = modem.pskdemod('M', 4, 'PhaseOffset', pi/4, 'SymbolOrder', 'Gray',
'OutputType', 'Bit');
   CRCgen       = crc.generator([1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
   CRCdet       = crc.detector  ([1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
end
%% Processing loop
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u           = randi([0 1], FRM,1);                    % Generate bit payload
   u1          = generate(CRCgen,u);                     % CRC insertion
   u2          = u1((end-C+1):end);                      % Tail-biting convolutional coding
   [~, state]  = convenc(u2,trellis);
   u3          = convenc(u1,trellis,state);
   u4          = fcn_RateMatcher(u3, L, codeRate);   % Rate matching
   u5          = fcn_Scrambler(u4, nS);                  % Scrambling
   u6          = modulate(Modulator, u5);               % Modulation
   u7          = TransmitDiversityEncoder2(u6);         % MIMO Alamouti encoder
```

```
    % Channel
    [u8, h8]   = MIMOFadingChan(u7);                        % MIMO fading channel
    sigpower = 10*log10(real(var(u8(:))));
    u9          = awgn(u8,snr,sigpower,'dB');
    % Receiver
    uA          =  TransmitDiversityCombiner3(u9, h8); % MIMO Alamouti combiner
    uB          = demodulate(Demodulator,uA);          % Demodulation
    uC          = fcn_Descrambler(uB, nS);              % Descrambling
    uD          = fcn_RateDematcher(uC, L);             % Rate de-matching
    uE          = [uD;uD];                              % Tail-biting
    uF          = vitdec(uE ,trellis,34,'trunc','hard'); % Viterbi decoding
    uG          = uF(Index);
    y           = detect(CRCdet, uG );                  % CRC detection
    numErrs   = numErrs + sum( y~=u );                % Update number of bit errors
    numBits   = numBits + FRM;
    nS          = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;
bits=numBits;
```

我们考察 Alamouti _ Encoder2 可以看到，函数首先根据输入的大小初始化输出，随后将输入进行变换并在为输出矩阵预留的位置上插入输入样本。而且，更新的 Alamouti _ Encoder2 不仅进行了预分配，也进行了向量化操作，而原来的 Alamouti _ Encoder1 仅仅是一个标量函数。Alamouti _ Encoder1 的主要问题在于首先初始化一个空矩阵，然后用 for 循环处理输出矩阵，使得每次循环都增大了输出矩阵的大小。这中频繁的动态内存分配会造成性能的劣化。

MATLAB function

<table>
<tr><td>

```
function [ber, bits]=zPDCCH_v3(...)
.
.
u7 = TransmitDiversityEncoder1(u6);
.
.
end
```

</td><td>

```
function [ber, bits]=zPDCCH_v4(...)
.
.
u7 = TransmitDiversityEncoder2(u6);
.
.
end
```

</td></tr>
<tr><td>

```
function y = TransmitDiversityEncoder1(in)
% Alamouti Transmit Diversity Encoder
% Space-Frequency to Space-
  Time transformation
in(2:2:end) = -conj(in(2:2:end));
% STBC Alamouti
```

</td><td>

```
function y = TransmitDiversityEncoder2(in)
% Alamouti Transmit Diversity Encoder
% Space-Frequency to Space-
  Time transformation
in(2:2:end) = -conj(in(2:2:end));
% STBC Alamouti
```

</td></tr>
</table>

y = Alamouti_Encoder1(in);	y = Alamouti_Encoder2(in);
% Scale y = y/sqrt(2);	% Scale y = y/sqrt(2);

```
function y= Alamouti_Encoder1(u)
% Space-Time Block Encoder
Tx=2;
LEN=size(u,1);
idx1=1:Tx:LEN-1;
idx2=idx1+1;
% Alamouti Space-Time Block Encoder
%   G = [ s1      s2 ]
%       [ -s2*    s1*]
y=[];
for n=1:LEN/Tx
  G=[      u(idx1(n))      u(idx2(n));...
     -conj(u(idx2(n))) conj(u(idx1(n)))];
  y=[y;G];
end
```

```
function y= Alamouti_Encoder2(u)
% Space-Time Block Encoder
Tx=2;
LEN=size(u,1);
idx1=1:Tx:LEN-1;
idx2=idx1+1;
% Alamouti Space-Time Block Encoder
%   G = [ s1      s2 ]
%       [ -s2*    s1*]
y=complex(zeros(LEN,Tx));
y(idx1,1)=u(idx1);
y(idx1,2)=u(idx2);
y(idx2,1)=-conj(u(idx2));
y(idx2,2)=conj(u(idx1));
```

通过运行下面的 MATLAB 脚本，我们可以确认优化是否得到了速度的提升。其结果显示八次循环处理 1 百万比特数据共需 82.71s（见图 9.10）。

Algorithm

MATLAB script: zPDCCH_v4_test

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\nVersion 4: Vectorization + Preallocation\n\n');
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v4 (snr, MaxNumBits, MaxNumBits);
end
time_4=toc;
fprintf(1,'Version 4: Time to complete %d iterations = %6.4f (sec)\n',
MaxSNR, time_4);
```

这一系列的仿真提供了一个模板。我们从初始的基准算法用 MATLAB 代码实现了 Alamouti 编码器和合并算法。基准算法可以认为是文字描述的空-时区块编码的算法数学公式转写。基于标量操作的 MATLAB 代码不能最快执行算法。在大部分情况下，我们需要将转换操作序列为 MATLAB 语言特有的基于向量的描述形式。这意味算法需要进行代码向量化和预分配数据。

这些额外的优化改写了 MATLAB 代码。我们既可以耗费时间优化我们的代

```
Version 4: Vectorization + Preallocation

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 4: Time to complete 8 iterations = 82.7194 (sec)
```

图 9.10 第四个版本的算法：循环八次耗时

码，也可以通过使用各种 MATLAB 工具箱进行优化。MATLAB 工具箱本身即对仿真进行了优化。所有的 MATLAB 工具箱函数都基于预分配和向量化。不仅如此，如前文所讨论的，DSP 和通信系统工具箱提供了高效的算法组件，即系统对象。在下一节中，我们会通过使用一些通信系统工具箱中的系统对象，让我们的算法组件运行更快。

9.6.3 系统对象

使用系统对象可以为 MATLAB 代码提速，特别是针对信号处理和通信领域。系统对象为 MATLAB 面向对象的算法，可以从 MATLAB 工具箱如通信系统工具箱调用。通过使用系统对象，我们将声明（创建系统对象）从算法执行中解耦出来，只进行一次参数设置和初始化，从而更高效的执行循环运算。一个系统对象可以在循环外生成和配置，然后可以在循环内通过单步方法调用它。DSP 和通信系统工具箱内的大部分系统对象都是 MATLAB 可执行文件（MEX）。一个 MEX 文件代码编译自 C 代码。很多算法优化已经集成在 MEX 中，从而仿真速度可得到提升。

PDCCH 的第五个版本使用通信系统工具箱的系统对象实现 Alamouti 编码器（Alamouti _ EncoderS 函数）和 Alamouti 合并器（Alamouti _ CombinerS 函数）。

Algorithm

MATLAB function

```
function [ber, bits]=zPDCCH_v5(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2048;
M=4; k=log2(M); codeRate=1/3;
snr = EbNo + 10*log10(k) + 10*log10(codeRate);
trellis=poly2trellis(7, [133 171 165]);
L=FRM+24;C=6; Index=[L+1:(3*L/2) (L/2+1):L];
%% Initializations
```

```
persistent Modulator Demodulator CRCgen CRCdet
if isempty(Modulator)
   Modulator    = modem.pskmod('M', 4, 'PhaseOffset', pi/4, 'SymbolOrder', 'Gray',
'InputType', 'Bit');
   Demodulator = modem.pskdemod('M', 4, 'PhaseOffset', pi/4, 'SymbolOrder', 'Gray',
'OutputType', 'Bit');
   CRCgen       = crc.generator([1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
   CRCdet       = crc.detector  ([1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
end
%% Processing loop
numErrs = 0; numBits = 0; nS=0;
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u            = randi([0 1], FRM,1);                   % Generate bit payload
   u1           = generate(CRCgen,u);                    % CRC insertion
   u2           = u1((end-C+1):end);                     % Tail-biting convolutional coding
   [~, state]  = convenc(u2,trellis);
   u3           = convenc(u1,trellis,state);
   u4           = fcn_RateMatcher(u3, L, codeRate);  % Rate matching
   u5           = fcn_Scrambler(u4, nS);                % Scrambling
   u6           = modulate(Modulator, u5);              % Modulation
   u7           = TransmitDiversityEncoderS(u6);       % MIMO Alamouti encoder
   % Channel
   [u8, h8]    = MIMOFadingChan(u7);                    % MIMO fading channel
   sigpower = 10*log10(real(var(u8(:))));
   u9           = awgn(u8,snr,sigpower,'dB');
   % Receiver
   uA           =  TransmitDiversityCombinerS(u9, h8); % MIMO Alamouti combiner
   uB           = demodulate(Demodulator,uA);          % Demodulation
   uC           = fcn_Descrambler(uB, nS);              % Descrambling
   uD           = fcn_RateDematcher(uC, L);             % Rate de-matching
   uE           = [uD;uD];                                % Tail-biting
   uF           = vitdec(uE ,trellis,34,'trunc','hard'); % Viterbi decoding
   uG           = uF(Index);
   y            = detect(CRCdet, uG );                   % CRC detection
   numErrs  = numErrs + sum( y~=u );                    % Update number of bit errors
   numBits  = numBits + FRM;
   nS           = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = numErrs/numBits;
bits=numBits;
```

这个版本使用通信系统工具箱的系统对象实现 Alamouti 编码器和 Alamouti 合并器，构建。TransmitDiversityCombinerS 函数和 TransmitDiversityEncoderS 函数。

MATLAB function

```
function y = TransmitDiversityEncoderS(in)
%#codegen
% Alamouti Transmit Diversity Encoder
% Space-Frequency to Space-
  Time transformation
in = sqrt(2) * in;
% STBC Alamouti
y = Alamouti_EncoderS(in);
% Scale
y = y/sqrt(2);
```

```
function y = TransmitDiversityCombinerS(in,
  chEst)
%#codegen
% Alamouti Transmit Diversity Combiner
% Scale
in = sqrt(2) * in;
% STBC Alamouti
y =  Alamouti_DecoderS(in, chEst);
% Space-Frequency to Space-
  Time transformation
y(2:2:end) = -conj(y(2:2:end));
```

```
function y = Alamouti_EncoderS(u)
% STBCENC Space-Time Block Encoder
% Outputs the Space-
  Time block encoded matrix
persistent hTDEnc;
if isempty(hTDEnc)
  % Use same object for either scheme
  hTDEnc = comm.OSTBCEncoder
  ('NumTransmitAntennas', 2);
end
% Alamouti Space-Time Block Encoder
y = step(hTDEnc, u);
```

```
function s = Alamouti_DecoderS(u,H)
%#codegen
% STBC_DEC STBC Combiner
persistent hTDDec
if isempty(hTDDec)
  hTDDec= comm.OSTBCCombiner(...
    'NumTransmitAnten-
  nas',2,'NumReceiveAntennas',2);
end
s = step(hTDDec, u, H);
```

注意，我们只在第一次访问函数时创建 comm. OSTBCEncoder 和 comm. OSTBC-Combiner 系统对象。这一过程通过标识系统对象为 MATLAB persistent 变量完成。我们随后使用 isempty 函数，以确保第一次运行时持久类型的变量为空，或者说并不进行初始化。随后通过调用相应系统对象的 step 函数执行两个 Alamouti 算法。

让我们现在验证通过使用系统对象，我们如何绕过预分配和向量化得到更快的执行速度。运行下面的 MATLAB 脚本调用使用系统对象的第五个版本算法。其结果（如图 9. 11 所示）显示八次循环处理 1 百万比特数据共需 81. 91s。这一结果与第四个版本算法相近。注意我们通过使用工具箱的系统对象绕过了改写代码的工作。

Algorithm

MATLAB script: zPDCCH_v4_test

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\nVersion 5: Using System objects for MIMO \n\n');
```

```
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v5 (snr, MaxNumBits, MaxNumBits);
end
time_5=toc;
fprintf(1,'Version 5: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_5);
```

```
Version 5: System objects for MIMO

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 5: Time to complete 8 iterations = 81.9175 (sec)
```

图 9.11　第五个版本的算法：循环八次耗时

为了对到目前为止讨论过的各个技术算法的加速程度进行记录，我们设计了一个辅助器 MATLAB 函数（Report _ Timing _ Results. m）。这个函数有四个输入：算法版本、基准算法耗时、当前算法耗时和优化算法的文字描述。它输出一个仿真耗时记录表。

Algorithm

MATLAB function: Report_Timing_Results

```
function y=Report_Timing_Results(M,a,b,str)
persistent Results
if isempty(Results)
   Results={};
end
Results(M).name=str;
Results(M).elapsed_time=b;
Results(M).acceleration=a/b;
disp('-------------------------------------------------------------------------------------');
disp('Versions of the Transceiver                    | Elapsed Time (sec)| Acceleration Ratio');
for m=1:M
fprintf(1,'%d. %-49s| %17.4f | %12.4f\n',m, Results(m).name, Results(m).elapsed_time,
Results(m).acceleration);
end
disp('-------------------------------------------------------------------------------------');
y=Results;
end
```

通过运行这个函数，我们记录每个不同版本算法的执行时间并对比基准算法计算加速率。结果显示使用系统对象加速仿真有 5.02 的加速率（见图 9.12）。

Versions of the Transceiver	Elapsed Time (sec)	Acceleration Ratio
1. Baseline	411.3030	1.0000
2. Vectorization	326.5071	1.2597
3. Vectorization along larger dimension	175.8478	2.3390
4. Vectorization + Preallocation	82.7194	4.9723
5. System objects for MIMO	81.9175	5.0209

图 9.12　执行时间和五个版本算法的加速率

Algorithm

MATLAB script

```
Report_Timing_Results(1,time_1,time_1,'Baseline');
Report_Timing_Results(2,time_1,time_2,'Vectorization');
Report_Timing_Results(3,time_1,time_3,'Vectorization along larger dimension');
Report_Timing_Results(4,time_1,time_4,'Vectorization + Preallocation');
Report_Timing_Results(5,time_1,time_5,'System objects for MIMO');
```

分析第五个版本的算法可以帮助我们找到优化的下一个瓶颈。MATLAB 分析器命令和报告如图 9.13 所示。

Algorithm

MATLAB script

```
profile on;
ber= zPDCCH_v5(snr, MaxNumBits, MaxNumBits);
profile viewer;
```

Profile Summary

Generated 15-Dec-2012 17:03:43 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
zPDCCH_v5	1	1.490 s	0.024 s	
MIMOFadingChan	49	0.503 s	0.009 s	
Channel.filter	49	0.493 s	0.057 s	
vitdec	49	0.299 s	0.046 s	
Channel.reset	49	0.281 s	0.046 s	
vit (MEX-file)	49	0.243 s	0.243 s	

图 9.13　第五个版本算法的分析总结报告

分析报告指出两个算法——MIMO 信道模型（MIMOFadingChan. m）和 Viterbi 译码器（vitdec 函数）——为下一个瓶颈。这两个算法基于通信系统工具箱中的两个函数：minochan 和 vitdec。这两个函数如同其他 MATLAB 工具箱中的函数一样，都经过了向量化和预分配优化。不过，将它们替换为相应的系统对象会在性能上得到更多提升。两个方面的改进决定了使用系统对象的优势：避免了反复的参数校验，以及使用 MATLAB MEX 执行。

9.6.3.1 MATLAB MEX 执行

函数 MIMOFadingChan. m 是 PDCCH 算法优化第五个版本的瓶颈。观察 MIMOFAdingChan 函数可以发现，其使用通信系统工具箱中的 minochan 对象实现 MIMO 信道滤波操作。使用持久类型变量，可使 minochan 对象只在函数第一次访问它时初始化。当我们每次调用函数时，我们执行对象的滤波器方法即得到信道模型的滤波输出（变量 y）和信道增益（变量 k）。

在第六个版本的算法中（见图 9.14），我们用系统对象 comm. MOMPChannel 实现 MIMO 信道滤波。考察这个 MIMOFadingChanS 函数，我们注意到当且仅当函数第一次访问时系统对象 comm. MIMOChannel 进行初始化。执行单步方法可以得到滤波输出和信道增益。

MATLAB function

function [ber, bits]=zPDCCH_v5(...) . .	**function [ber, bits]=zPDCCH_v6(...)** . .
[u8, h8] = MIMOFadingChan(u7);	[u8, h8] = MIMOFadingChanS(u7);
. . end	. . **end**

```
function [y, h] = MIMOFadingChan(in)
% MIMOFadingChan
numTx=2;
numRx=2;
chanSRate=(2048*15000);
Doppler=70;
PathDelays = 0;
PathGains  = 0;
persistent chanObj
if isempty(chanObj)
  chanObj = mimochan(numTx,numRx,
(1/chanSRate),Doppler,PathDelays,
  PathGains);
  chanObj.NormalizePathGains = 1;
```

```
function [y, h] = MIMOFadingChanS(in)
% MIMOFadingChan
numTx=2;
numRx=2;
chanSRate=(2048*15000);
Doppler=70;
PathDelays = 0;
PathGains  = 0;
persistent chanObj
if isempty(chanObj)
  chanObj = comm.MIMOChannel
  ('SampleRate', chanSRate,
      'MaximumDopplerShift', Doppler,
      'PathDelays', PathDelays,
```

```
  chanObj.StorePathGains = 1;
  chanObj.ResetBeforeFiltering = 1;
end
y         = filter(chanObj, in);
ChGains = chanObj.PathGains;
Len       = size(in,1);
h    = complex(zeros(Len,numTx,numRx));
h(:) = ChGains(:,1,:,:);
```

```
          'AveragePathGains', PathGains,
          'NumTransmitAntennas', numTx,...
          'TransmitCorrelationMatrix',
  eye(numTx),...
          'NumReceiveAntennas', numRx,...
          'ReceiveCorrelationMatrix',
  eye(numRx),...
          'PathGainsOutputPort', true,...
          'NormalizePathGains', true,...
          'NormalizeChannelOutputs', true);
end
[y, G] = step(chanObj, in);
Len       = size(in,1);
PathG     = com-
  plex(zeros(Len,numTx,numRx));
PathG(:) = G(:,1,:,:);
h         = PathG;
```

我们很快发现，comm. MIMOChannel 系统对象和 minochan 对象有很多相似之处。系统对象基于 MATLAB MEX 执行（编译的 C 代码）并进行了多种优化。我们希望通过使用 comm. MIMOChannel 系统对象的 MIMOFadingChanS 函数得到性能提升。为了验证这一点，我们运行下面的 MATLAB 脚本，对比到目前为止各个版本算法的性能（见图 9. 15）。

```
Version 6: System objects for MIMO & Channel modeling

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 6: Time to complete 8 iterations = 59.9528 (sec)
```

图 9. 14　第六个版本的算法：循环八次耗时

Versions of the Transceiver	Elapsed Time (sec)	Acceleration Ratio
1. Baseline	411.3030	1.0000
2. Vectorization	326.5071	1.2597
3. Vectorization along larger dimension	175.8478	2.3390
4. Vectorization + Preallocation	82.7194	4.9723
5. System objects for MIMO	81.9175	5.0209
6. System objects for MIMO & Channel	59.9528	6.8604

图 9. 15　执行时间和六个版本算法的加速率

Algorithm

MATLAB script: zPDCCH_v6_test

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\nVersion 6: System objects for MIMO & Channel\n\n');
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v6 (snr, MaxNumBits, MaxNumBits);
end
time_6=toc;
fprintf(1,'Version 6: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_6);
Report_Timing_Results(6,time_1,time_6, System objects for MIMO & Channel);
```

我们下面分析第六个版本的算法。如分析总结报告所示，通信系统工具箱的Viterbi 译码器函数 vitdec 是第六版算法的瓶颈（见图 9. 16）。

9. 6. 3. 2 避免反复的参数校验

第七版算法（见图 9. 17）通过替换 Viterbi 译码器函数 vitdec 为相应的系统对象 comm. ViterbiDecoder 进行优化。使用系统对象可以避免反复的参数校验而提升速度。通过使用系统对象，我们将声明（创建系统对象）从算法执行中解耦出来，只在 While 循环外进行一次参数设置和初始化。但是，在 vitdec 函数中，每次循环调用函数时，程序都对如网格结构以及终止和判决方法等参数进行校验并在主程序调用之前生成合适的中间变量。

当我们尝试不同函数模式以及与命令行交互操作时，这类参数处理是必须进行的。不过，当函数的参数确定且循环执行时，避免额外的参数处理——如系统对象所设计的——可以提高仿真速度。

Profile Summary

Generated 18-Dec-2012 13:26:20 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
zPDCCH_v6	1	1.162 s	0.011 s	
vitdec	49	0.351 s	0.041 s	
vit (MEX-file)	49	0.310 s	0.310 s	

图 9. 16 第六个版本算法的分析总结报告

```
Version 7: System objects for MIMO & Channel & Viterbi

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 7: Time to complete 8 iterations = 58.8660 (sec)
```

图 9.17 第七个版本的算法：循环八次耗时

MATLAB function

```
function [ber, bits]=zPDCCH_v6(...)
.
.
while ((numErrs < maxNumErrs) &&
   (numBits < maxNumBits))
.
uF        = vitdec(uE,trellis,34,'trunc','hard');
   % Viterbi decoding
.
.
end
```

```
function [ber, bits]=zPDCCH_v7(...)
.
Viterbi=comm.ViterbiDecoder(
'TrellisStructure', trellis, 'InputFormat','Hard',
'TerminationMethod','Truncated');
.
while ((numErrs < maxNumErrs) &&
   (numBits < maxNumBits))
.
uF      = step(Viterbi, uE); % Viterbi decoding
.
.
end
```

我们可以通过运行下面的 MATLAB 脚本验证优化效果，脚本对比第七个版本和前六个算法的性能提升情况（见图 9.18）。

Algorithm

MATLAB script: zPDCCH_v7_test

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\nVersion 7: System objects for MIMO & Channel & Viterbi\n\n');
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v7 (snr, MaxNumBits, MaxNumBits);
end
time_7=toc;
fprintf(1,'Version 7: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR,
time_7);
Report_Timing_Results(7,time_1,time_7,'System objects for MIMO & Channel
& Viterbi');
```

```
Versions of the Transceiver                            | Elapsed Time (sec)| Acceleration Ratio
 1. Baseline                                           |        411.3030 |        1.0000
 2. Vectorization                                      |        326.5071 |        1.2597
 3. Vectorization along larger dimension               |        175.8478 |        2.3390
 4. Vectorization + Preallocation                      |         82.7194 |        4.9723
 5. System objects for MIMO                            |         81.9175 |        5.0209
 6. System objects for MIMO & Channel                  |         59.9528 |        6.8604
 7. System objects for MIMO & Channel & Viterbi        |         58.8660 |        6.9871
```

图 9.18 执行时间和七个版本算法的加速率

9.6.3.3 使用所有可用的系统对象

在第八个版本（见图 9.19）中，我们使用和 PDCCH 算法有关的所有的系统对象。除了到目前为止使用的系统对象之外，我们也使用了调制器、解调器、两个卷积编码器（尾比特编码），和 CRC 生成及校验组件。

```
Version 8: Using All available System objects

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 8: Time to complete 8 iterations = 48.9014 (sec)
```

图 9.19 第八个版本的算法：循环八次耗时

Algorithm

MATLAB function

```
function [ber, bits]=zPDCCH_v8(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2048;
M=4; k=log2(M); codeRate=1/3;
snr = EbNo + 10*log10(k) + 10*log10(codeRate);
trellis=poly2trellis(7, [133 171 165]);
L=FRM+24;C=6; Index=[L+1:(3*L/2) (L/2+1):L];
%% Initializations
persistent Modulator AWGN DeModulator BitError ConvEncoder1 ConvEncoder2 Viterbi
CRCGen CRCDet
if isempty(Modulator)
   Modulator      = comm.QPSKModulator('BitInput',true);
   AWGN           = comm.AWGNChannel('NoiseMethod', 'Variance', 'VarianceSource',
'Input port');
```

```
   DeModulator =  comm.QPSKDemodulator('BitOutput',true);
   BitError        = comm.ErrorRate;
   ConvEncoder1=comm.ConvolutionalEncoder('TrellisStructure', trellis,
'FinalStateOutputPort', true, ...
       'TerminationMethod','Truncated');
   ConvEncoder2 =  comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'InitialStateInputPort', true,...
       'TrellisStructure', trellis);
   Viterbi=comm.ViterbiDecoder('TrellisStructure', trellis,
'InputFormat','Hard','TerminationMethod','Truncated');
   CRCGen = comm.CRCGenerator('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
   CRCDet = comm.CRCDetector   ('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
end
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
results=zeros(3,1);
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u          = randi([0 1], FRM,1);                   % Generate bit payload
   u1         = step(CRCGen, u);                        % CRC insertion
   u2         = u1((end-C+1):end);                     % Tail-biting convolutional coding
   [~, state]  = step(ConvEncoder1, u2);
   u3         = step(ConvEncoder2, u1,state);
   u4         = fcn_RateMatcher(u3, L, codeRate);  % Rate matching
   u5         = fcn_Scrambler(u4, nS);                 % Scrambling
   u6         = step(Modulator, u5);                  % Modulation
   u7         = TransmitDiversityEncoderS(u6);         % MIMO Alamouti encoder
    % Channel
   [u8, h8]    = MIMOFadingChanS(u7);                   % MIMO fading channel
   noise_var = real(var(u8(:)))/(10.^(0.1*snr));
   u9         = step(AWGN, u8, noise_var);              % AWGN
   % Receiver
   uA          =  TransmitDiversityCombinerS(u9, h8);% MIMO Alamouti combiner
   uB          = step(DeModulator, uA);                % Demodulation
   uC          = fcn_Descrambler(uB, nS);               % Descrambling
   uD          = fcn_RateDematcher(uC, L);              % Rate de-matching
   uE          = [uD;uD];                               % Tail-biting
   uF          = step(Viterbi, uE);                     % Viterbi decoding
   uG          = uF(Index);
   y          = step(CRCDet, uG );                       % CRC detection
   results     = step(BitError, u, y);                  % Update number of bit errors
   numErrs     = results(2);
   numBits     = results(3);
   nS          = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = results(1); bits= results(3);
reset(BitError);
```

通过运行下面的 MATLAB 脚本，对比第八个版本和前七个算法的性能提升情况（见图 9. 20），我们可以看到使用所有有关的系统对象可以得到仿真速度上的提升。

```
Versions of the Transceiver                             | Elapsed Time (sec)| Acceleration Ratio
 1. Baseline                                            |          411.3030 |        1.0000
 2. Vectorization                                       |          326.5071 |        1.2597
 3. Vectorization along larger dimension                |          175.8478 |        2.3390
 4. Vectorization + Preallocation                       |           82.7194 |        4.9723
 5. System objects for MIMO                             |           81.9175 |        5.0209
 6. System objects for MIMO & Channel                   |           59.9528 |        6.8604
 7. System objects for MIMO & Channel & Viterbi         |           58.8660 |        6.9871
 8. System objects for all                              |           48.9014 |        8.4109
```

图 9. 20　执行时间和八个版本算法的加速率

Algorithm

MATLAB script: zPDCCH_v8_test

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\nVersion 8: Using All available System objects\n\n');
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v8(snr, MaxNumBits, MaxNumBits);
end
time_8=toc;
fprintf(1,'Version 8: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_8);
Report_Timing_Results(8,time_1,time_8,'System objects for all');
```

我们已经展示了如何编写更好的 MATLAB 代码可以更快仿真。我们也展示了使用通信系统工具箱的系统工具可以提升算法仿真速度（在大部分情况下）。其他使用系统对象的好处在于其支持 MATLAB 代码转换器进行 MATLAB - C 代码转换。这一加速特性将在下面进行讨论。

9. 7　使用加速功能

我们前面主要关注 MATLAB 程序的优化。除了代码优化之外，性能提升也可以通过其他运算工具如编译 C 代码设计实现。MATLA 并行计算产品也可发挥多核处理、计算机束，以及 GPU 的优势。MATLAB 代码转换器可以自动将 MATLAB 代码转换为 C 代码，然后进行编译以提高仿真速度。在下一节中，我们将介绍这些特性以得到更高的仿真速度。

9.7.1 MATLAB—C 代码生成

将 MATLAB 代码替换为自动生成的 MEX（函数）可以提高仿真速度。使用 MATLAB 代码转换器，我们可以生成可读和可移植的 C 代码，并将其编译为 MEX 函数，替换现有 MATLAB 算法中相应的部分。加速的效果依赖于算法。测定加速效果的最好办法是使用 MATLAB 代码转换器生成 MEX 并测试加速情况。假如算法内存在单精度数据类型、定点数据类型、循环 - 分支，或未被向量化的代码，我们就可以观察到明显的加速。很多 MATLAB 语言和工具箱，包括通信系统工具箱，都支持代码转换。

现在，我们将 PDCCH 算法的第八个版本转换为 MEX 函数。这个版本的算法为第九个版本的加速算法。这一处理可通过一行 MATLAB 代码转换器命令（codegen）完成。下面的 MATLAB 脚本为如何调用 codegen 命令将函数 zPDCCH _ v8. m 转换为 C 代码，并进行编译生成 MEX 函数。假如我们不重命名输出函数，则其默认后缀为_ mex，即 zPDCCH _ v8 _ mex。

Algorithm

MATLAB script: zPDCCH_v8_codegen

```
MaxSNR=8;
MaxNumBits=1e5;
fprintf(1,'\n\nGenerating MEX function for zPDCCH_v8.m \r');
codegen -args { MaxSNR, MaxNumBits, MaxNumBits } zPDCCH_v8.m
fprintf(1,'Done.\r');
```

通过下面的 MATLAB 脚本我们可以验证优化工作是否有效。八次循环处理 1 百万比特数据共需约 37.18s（见图 9.21 和图 9.22）

Algorithm

MATLAB script: zPDCCH_v9_test

```
MaxSNR=8;
MaxNumBits=1e6;
fprintf(1,'\nVersion 9: MATLAB to C code generation (MEX)\n\n');
tic;
for EbNo=1:MaxSNR
   fprintf(1,'Iteration number %d\r',EbNo);
   ber= zPDCCH_v8_mex(snr, MaxNumBits, MaxNumBits);
end
time_9=toc;
fprintf(1,'Version 9: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_9);
```

结果显示 MEX 版算法仿真的结果显示，当编译系统对象为 MEX 函数后，其

仿真速度比前几个版本有所提高。

```
Version 9: MATLAB to C code generation (MEX)

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 9: Time to complete 8 iterations = 31.2941 (sec)
```

图 9.21　第九个版本的算法：循环八次耗时

```
Versions of the Transceiver                              | Elapsed Time (sec)| Acceleration Ratio
 1. Baseline                                             |         411.3030 |        1.0000
 2. Vectorization                                        |         326.5071 |        1.2597
 3. Vectorization along larger dimension                 |         175.8478 |        2.3390
 4. Vectorization + Preallocation                        |          82.7194 |        4.9723
 5. System objects for MIMO                              |          81.9175 |        5.0209
 6. System objects for MIMO & Channel                    |          59.9528 |        6.8604
 7. System objects for MIMO & Channel & Viterbi          |          58.8660 |        6.9871
 8. System objects for all                               |          48.9014 |        8.4109
 9. Version 8 + MATLAB to C code generation (MEX)        |          31.4722 |       13.0688
```

图 9.22　执行时间和九个版本算法的加速率

如我们预期，使用 MATLAB—C 代码生成可以加速仿真。虽然使用系统对象的算法已经得到高度优化，但代码生成可以通过锁定变量大小和数据类型的方法加速仿真。这一处理通过移除了对每一行代码检查大小和数据类型的解释语言，而使执行更有效率。假如一个算法包括隐式多线程计算，函数会调用 IPP 或 BLAS 库，内建函数对 MATLAB 在 PC 上的执行进行优化（如快速傅里叶变换），或进行代码向量化。在这种情况，我们将不会看到明显的速度提升。

9.7.2　并行运算

通过使用平行计算工具箱，我们可以在一个多核平台运行多个 MATLAB 线程（MATLAB 计算引擎）。仿真可以通过分配多个 MATLAB 线程得到提速。这一过程更多的使用并行计算而较少的使用隐式多线程。这一方法多用于参数扫描应用和蒙特卡洛分析的仿真。另外，MATLAB 平台的并行处理应用运行在单计算机、束或计算网格上。

并行计算工具箱也提供高级编程组件如 parfor 命令。通过使用 parfor，我们可以将 for 循环分配到多个 MATLAB 线程中执行。为了使用 parfor，循环迭代必

须独立，即不存在嵌套循环。假如我们想加速嵌套循环或分支结构的循环，我们就需要考虑优化 For 循环体代码的同时使用 C 代码转换。因为 parfor 循环会带来多 MATLAB 线程间的通信消耗，所以当我们只进行简单小数量仿真时，parfor 可能并没有明显的效果。

现在，我们调用 MEX 函数和 parfor 循环表示 PDCCH 算法的第九个版本。在这之前，我们必须访问计算机多核。matlabpool 命令（或最新 MATLAB 版本中的 parpool 命令）可以使我们访问计算机的多核并分配 MATLAB 线程。

Algorithm

MATLAB script: zPDCCH_vA_test

```
isOpen = matlabpool('size') > 0;
if ~isOpen
   fprintf(1,'Parallel Computing Toolbox is starting ...\n');
   matlabpool;
end
```

我们可以用 parfor 循环替代 for 循环进行并行处理，如下面的 MATLAB 脚本所示。

Algorithm

MATLAB script: zPDCCH_vA_test

```
MaxSNR=8;
MaxNumBits=1e6;
fprintf(1,'\nVersion 10: Parallel computing (parfor) + MEX \n\n');
tic;
parfor snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   ber= zPDCCH_v9(snr, MaxNumBits, MaxNumBits);
end
time_A=toc;
fprintf(1,'Version 10: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_A);
```

我们可以观察到通过使用通信系统工具箱系统对象和 MATLAB - C 代码转换，以及并行处理，我们可以将运算耗时减小到约 18. 38s（见图 9. 23）。其为基准版本的 22. 36 倍，是第五个版本——即用基础 MATLAB 编程方法——的 3. 45 倍（见图 9. 24）。如很多性能标准，加速的效果取决于算法、安装 MATLAB 的平台、生成 MEX 所需的 C/C ++ 编译器和计算机处理器为几个核。

```
Version 10: Parallel computing (parfor) + MEX

Iteration number 6
Iteration number 5
Iteration number 4
Iteration number 7

Iteration number 3
Iteration number 2
Iteration number 1
Iteration number 8

Version 10: Time to complete 8 iterations = 18.3899 (sec)
```

图 9.23 第十个版本的算法：循环八次耗时

```
Versions of the Transceiver                           | Elapsed Time (sec)| Acceleration Ratio
 1. Baseline                                          |          411.3030 |       1.0000
 2. Vectorization                                     |          326.5071 |       1.2597
 3. Vectorization along larger dimension              |          175.8478 |       2.3390
 4. Vectorization + Preallocation                     |           82.7194 |       4.9723
 5. System objects for MIMO                           |           81.9175 |       5.0209
 6. System objects for MIMO & Channel                 |           59.9528 |       6.8604
 7. System objects for MIMO & Channel & Viterbi       |           58.8660 |       6.9871
 8. System objects for all                            |           48.9014 |       8.4109
 9. Version 8 + MATLAB to C code generation (MEX)     |           31.4722 |      13.0688
10. Version 8 + MEX + Parallel computing (parfor)     |           18.3899 |      22.3657
```

图 9.24 执行时间和十个版本算法的加速率

9.8 使用 Simulink 模型

到目前为止，我们对 MATLAB 程序进行优化获得了更好的性能。针对 Simulink 模型描述的算法也可以进行相同的工作。Simulink 可以将设计以区块图表示。这种图形化表示方法可以表示架构和层次并易于理解。

通信系统工具箱的算法既可以以系统对象的形式在 MATLAB 程序中使用，也可以以组件的形式在 Simulink 中使用。例如，第八个版本的 PDCCH 算法，使用多个通信系统工具箱的系统对象。在本节中我们在 Simulink 模型中实现相同的算法。我们将首先验证其数值性能是否与 MATLAB 程序相同，然后考察各种可以提速仿真的 Simulink 优化技术。

9.8.1 创建 Simulink 模型

图 9.25 为 Simulink 模型（zPDCCH _v8 _default. xls）表示的第八版 PDCCH

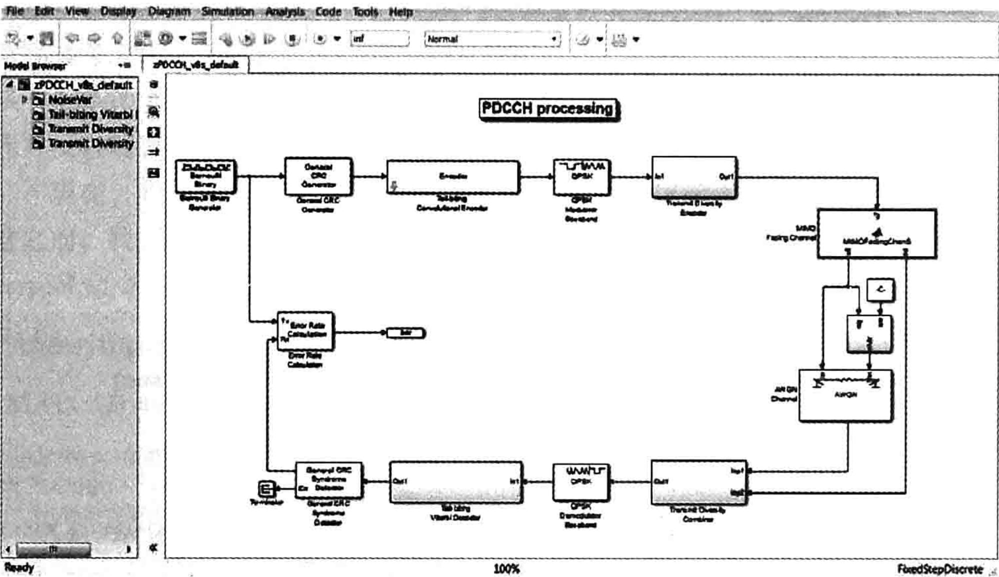


图 9.25　Simulink 模型表示 PDCCH 算法

算法。因 MATLAB 实现使用了系统对象，所以将 MATLAB 程序转换为 Simulink 的处理很简单。从通信系统工具箱的组件库中我们可以调用如卷积编码器、Viterbi 译码器等等的组件。从给定系统工具箱中调用的组件数量和属性与系统对象相同。因此，我们可以通过从系统对象中拷贝属性设置，轻松完成在 Simulink 环境下的组件属性设置。对算法上如发射分集这些涉及系统对象和一些基本 MATLAB 代码的部分，我们建立进行相同工作的 Simulink 子系统组件。对算法组件中包含若干子组件或子系统而不能简单实现的，我们可以用 Simulink 的 MATLAB 函数组件直接将 MATLAB 函数转换为 Simulink 组件。我们用这一方法实现 MIMO 衰落信道组件。

9.8.2　验证数值等价性

我们用 bertool 验证 MATLAB 实现和 Simulink 是否有相同的数值结果。bertool 可以进行如下操作：

1）循环遍历 Eb/N0 值；

2）代入每一个值执行 Simulink 模型或 MATLAB 函数；

3）通过两个参数将仿真终止条件传递给误码计算组件：最大误码数和最大处理比特数；

4）记录当前循环所得的 BER 值并描出 BER 曲线。

如图 9.26 所示，我们同时处理算法第八个版本的 MATLAB 函数（zPDCCH _

v8. m）和 Simulink（zPDCCH _ v8. default. xls）。通过遍历从 0 到 4 每个 SNR 值，并以分辨率 1/2 计算，以及设定最大误码数和最大处理比特数为 1 千万，我们可以比较 BER 随 SNR 值变化的曲线。如图 9. 27 所示，两者的数值结果非常接近。在较大 SNR 值时，数值结果有些许偏差。其原因为我们只设定对每个 SNR 值只仿真 1 千万比特。高 SNR 的 BER 在 1e－6 到 1e－7 量级。一点点的误码比特就会对性能造成影响。当设定更大数量的比特数时，仿真的数值结果完全一致。

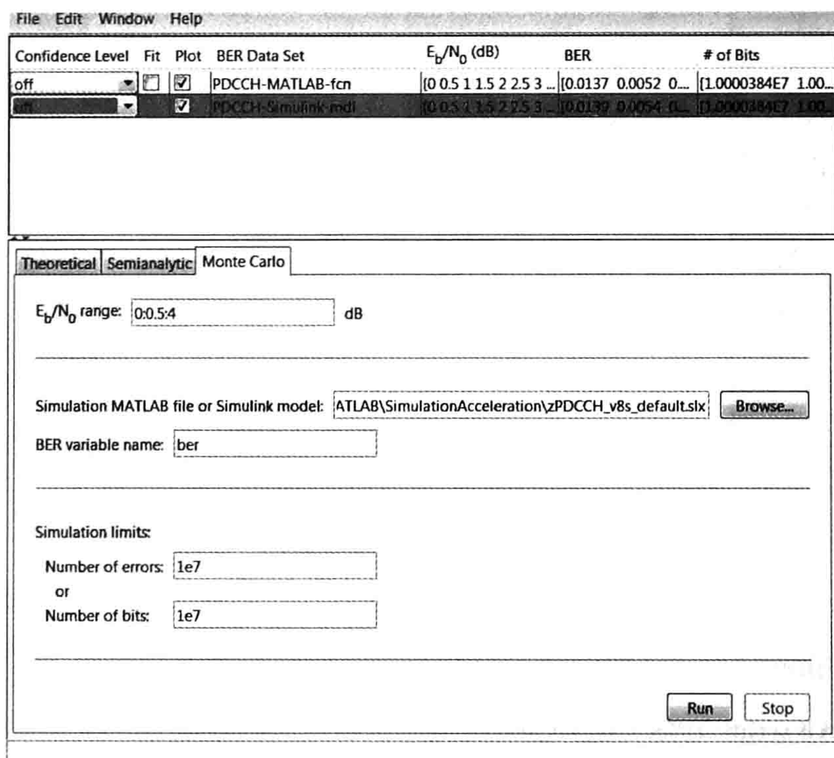


图 9. 26　bertool 遍历 PDCCH MATLAB 和 Simulink 模型

现在我们可以确认两种算法有数值上的等价性，下面让我们比较两者的运算耗时。

9. 8. 3　Simulink 基准模型

现在，我们运行如下 MATLAB 脚本，使用 sim 命令运行基准 Simulink 模型。仿真基准 Simulink 模型耗时约 84. 59s 处理在八个循环中 1 百万比特数据（见图 9. 28 和图 9. 29）

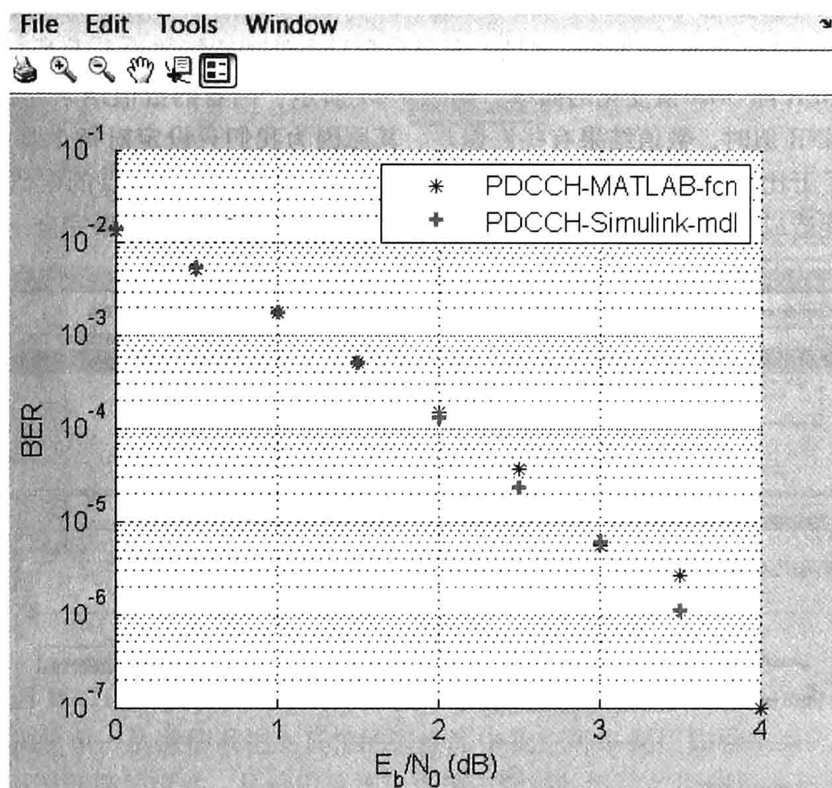


图 9.27 BER 曲线：PDCCH 算法的 MATLAB 和 Simulink 实现

Algorithm

MATLAB script: zPDCCH_vB_test

```
MaxSNR=8;
MaxNumBits=1e6;
fprintf(1,'\nVersion 11: Version 8 Simulink normal mode'\n\n');
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   sim('zPDCCH_v8s_default');
end
time_11=toc;
fprintf(1,'Version 11: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_11);
```

9.8.4 优化 Simulink 模型

我们可以通过多种方法优化 Simulink 仿真速度，如关闭可视化和调试，以及

```
Version 11: Version 8 Simulink normal mode

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 11: Time to complete 8 iterations = 84.5959 (sec)
```

图 9.28　第十一个版本的算法：循环八次耗时

```
Versions of the Transceiver                            | Elapsed Time (sec)| Acceleration Ratio
 1. Baseline                                           |        411.3030 |        1.0000
 2. Vectorization                                      |        326.5071 |        1.2597
 3. Vectorization along larger dimension               |        175.8478 |        2.3390
 4. Vectorization + Preallocation                      |         82.7194 |        4.9723
 5. System objects for MIMO                            |         81.9175 |        5.0209
 6. System objects for MIMO & Channel                  |         59.9528 |        6.8604
 7. System objects for MIMO & Channel & Viterbi        |         58.8660 |        6.9871
 8. System objects for all                             |         48.9014 |        8.4109
 9. Version 8 + MATLAB to C code generation (MEX)      |         31.4722 |       13.0688
10. Version 8 + MEX + Parallel computing (parfor)      |         18.3899 |       22.3657
11. Version 8 Simulink normal mode                     |         84.5959 |        4.8620
```

图 9.29　执行时间和十一个版本算法的加速率

引入我们在 MATLAB 编程中介绍过的加速功能，如 C 代码生成和并行计算等。我们将在本节中从 PDCCH 算法基准 Simulink 模型入手，逐一介绍这些优化技术。

9.8.4.1　仿真配置

最直接的加速方法即在仿真中关闭可视化和调试功能。Simulink 可以以多种模式执行一个模型，包括普通模式、加速器模式，和最快加速器模式。在普通模式下，默认配置参数被设定以一个有效的仿真模型增量装配帮助调试。这体现在每个模型仿真菜单的所有模型配置参数都可选。图 9.30 所示为 zPDCCH _ v8 _ default. slx 模型的默认配置参数，它们可在仿真目标选项卡中找到。

正如我们所看到的，与调试有关的属性，如“Enable debugging/animation”，“Enable overflow detection”，以及“Echo expression without semicolon”，默认为打开。一些可以帮助设计者在运行时检查语法错误的功能，如“Ensure memory integrity”和“Ensure responsiveness”也是默认打开的。显然，这会增加仿真开销。关闭它们能获得一定程度的加速。图 9.31 所示为去掉上述功能后新的仿真目标参数。

为了明确通过如上操作我们可以得到多少性能提升，我们运行以下 MATLAB

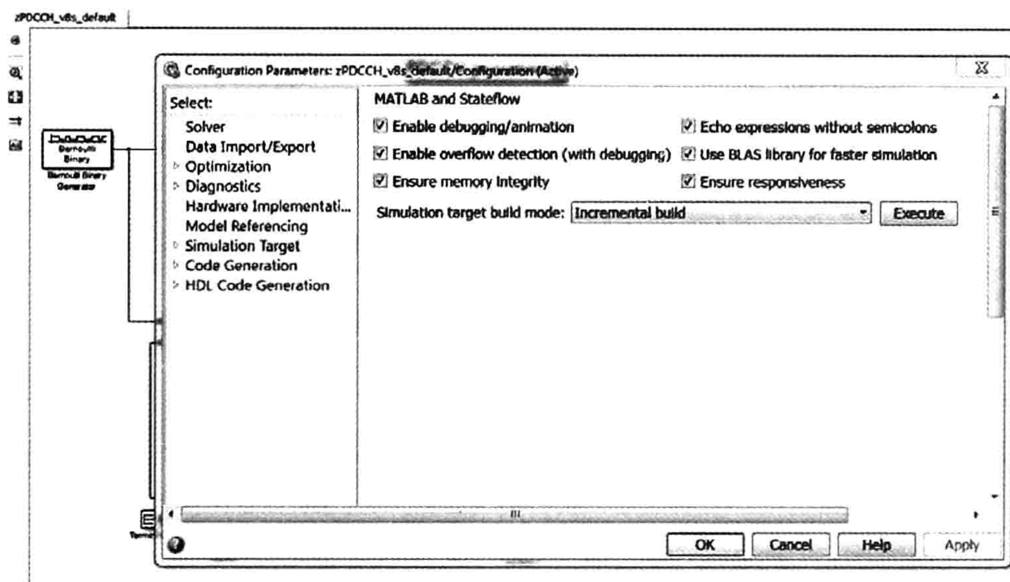


图 9. 30　模型配置参数：默认 – 普通模式

脚本，进行对普通模式进行更多优化后的仿真。其结果显示，仿真时间从原来的约 84s 减少到 44s（见图 9. 32）。可以看到，这个结果非常可观，其仿真速度可与第八个版本的 MATLAB 算法相媲美（见图 9. 33）。

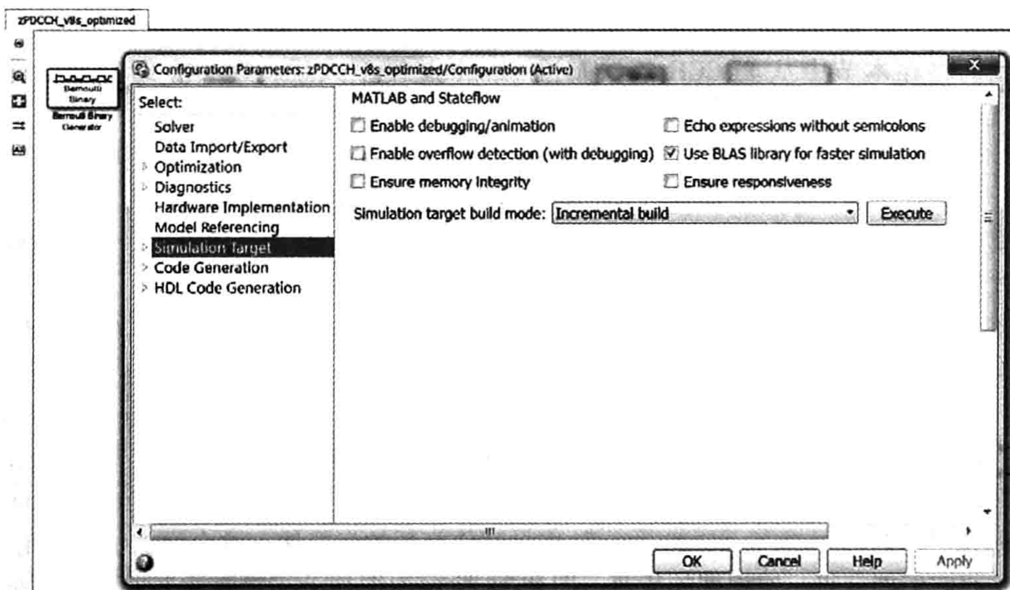


图 9. 31　模型配置参数：仿真目标

```
Version 12: Version 8 Simulink normal mode optimized

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 12: Time to complete 8 iterations = 44.3369 (sec)
```

图 9.32　第十二个版本的算法：循环八次耗时

Versions of the Transceiver	Elapsed Time (sec)	Acceleration Ratio
1. Baseline	411.3030	1.0000
2. Vectorization	326.5071	1.2597
3. Vectorization along larger dimension	175.8478	2.3390
4. Vectorization + Preallocation	82.7194	4.9723
5. System objects for MIMO	81.9175	5.0209
6. System objects for MIMO & Channel	59.9528	6.8604
7. System objects for MIMO & Channel & Viterbi	58.8660	6.9871
8. System objects for all	48.9014	8.4109
9. Version 8 + MATLAB to C code generation (MEX)	31.4722	13.0688
10. Version 8 + MEX + Parallel computing (parfor)	18.3899	22.3657
11. Version 8 Simulink normal mode	84.5959	4.8620
12. Version 8 Simulink normal mode optimized	44.3369	9.2768

图 9.33　执行时间和十二个版本算法的加速率

Algorithm

MATLAB script: zPDCCH_vC_test

```
MaxSNR=8;
MaxNumBits=1e6;
fprintf(1,'\nVersion 12: Version 8 Simulink normal mode optimized\n\n');
tic;
for EbNo=1:MaxSNR
   fprintf(1,'Iteration number %d\r',EbNo);
   sim('zPDCCH_v8s_optimized');
end
time_12=toc;
fprintf(1,'Version 12: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_12);
```

9.8.4.2　最快加速器模式

另一种直接加速的方式是使用仿真的最快加速器模式。如图 9.34 所示，仿真模式变为最快加速器模式可以轻松实现加速。最快加速器模式生成模型的 MEX 函数并执行编译后的代码。在这点上，最快加速器模式与从 MATLAB 函数

生成 MEX 文件类似。

当仿真第一次运行时，模型被编译生成 MEX 函数。对模型 zPDCCH _ v8s _ optimized. slx，MATLAB 工作区会出现如图 9. 35 所示信息并生成 MEX 文件 zPDCCH _ v8s _ optimized _ sfun。

通过运行下面的 MATLAB 脚本，我们在最快加速器模式下对优化后的模型进行仿真。结果显示仿真耗时 40s（见图 9. 36）。对比使用 MATLAB—C 代码转换的 MATLAB 函数（第九个版本的算法）结果，可以看出 MATLAB 函数使用 MEX 文件的仿真速度要更快一些（见图 9. 37）。在下面我们将会介绍一个简单的设置减小两者的差距。

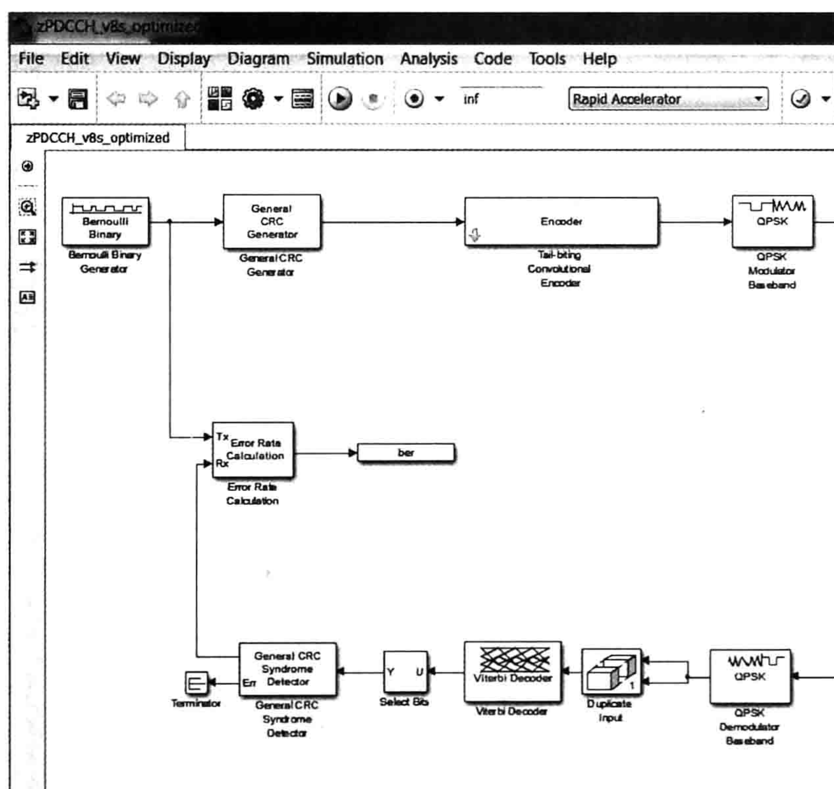


图 9. 34 在最快加速器模式下仿真

```
### Building the rapid accelerator target for model: zPDCCH_v8s_optimized
### Successfully built the rapid accelerator target for model: zPDCCH_v8s_optimized
```

图 9. 35 在最快加速器模式下编译一个 Simulink 模型

```
Version 13: Version 8 Simulink rapid accelerator

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 13: Time to complete 8 iterations = 40.5559 (sec)
```

图 9.36　第十三个版本的算法：循环八次耗时

```
Versions of the Transceiver                           | Elapsed Time (sec)| Acceleration Ratio
 1. Baseline                                          |          411.3030 |        1.0000
 2. Vectorization                                     |          326.5071 |        1.2597
 3. Vectorization along larger dimension              |          175.8478 |        2.3390
 4. Vectorization + Preallocation                     |           82.7194 |        4.9723
 5. System objects for MIMO                           |           81.9175 |        5.0209
 6. System objects for MIMO & Channel                 |           59.9528 |        6.8604
 7. System objects for MIMO & Channel & Viterbi       |           58.8660 |        6.9871
 8. System objects for all                            |           48.9014 |        8.4109
 9. Version 8 + MATLAB to C code generation (MEX)     |           31.4722 |       13.0688
10. Version 8 + MEX + Parallel computing (parfor)     |           18.3899 |       22.3657
11. Version 8 Simulink normal mode                    |           84.5959 |        4.8620
12. Version 8 Simulink normal mode optimized          |           44.3369 |        9.2768
13. Version 8 Simulink rapid accelerator              |           40.5559 |       10.1416
```

图 9.37　执行时间和十三个版本算法的加速率

Algorithm

MATLAB script: zPDCCH_vD_test

```
MaxSNR=8;
MaxNumBits=1e6;
fprintf(1,'\nVersion 13: Version 8 Simulink rapid accelerator\n\n');
tic;
for EbNo=1:MaxSNR
  fprintf(1,'Iteration number %d\r',EbNo);
  sim('zPDCCH_v8s_optimized','SimulationMode','rapid');
end
time_13=toc;
fprintf(1,'Version 13: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_13);
```

9.8.4.3　优化最快加速器模式

在最快加速器模式下，每当 Simulink 模型改变时 Simulink 都会重新生成 MEX 文件。Simulink 判断最快加速器可执行文件是否更新的时间远远小于其生

成代码的时间。我们可以利用这一特性测试设计折中。例如，我们可以生成一个最快加速器目标代码，并用其仿真不同 SNR 条件下模型的结果。这种不同条件的变化并不会重新生成目标代码。这使得这种模式非常高效。目标代码在第一次运行模型时生成，Simulink 在之后的运行中只会用必要的时间确定目标代码是否更新。我们甚至可以通过 sim 命令关闭更新检查而绕过最快加速器模式的更新检查循环：

Algorithm

```
sim(model_name,'SimulationMode','rapid','RapidAcceleratorUpToDateCheck', 'off')
```

为了验证最快加速器的优化是否有效，我们调用如下 MATLAB 测试脚本。如结果所示，优化后的速度几乎是优化前的两倍。我们可以观察到 Simulink 生成的代码比 MATLAB 代码（第九版）运行的更快（图 9.38 和图 9.39）。

```
Version 14: Version 8 Simulink rapid accelerator optimized

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 14: Time to complete 8 iterations = 22.2629 (sec)
```

图 9.38　第十四个版本的算法：循环八次耗时

Versions of the Transceiver	Elapsed Time (sec)	Acceleration Ratio
1. Baseline	411.3030	1.0000
2. Vectorization	326.5071	1.2597
3. Vectorization along larger dimension	175.8478	2.3390
4. Vectorization + Preallocation	82.7194	4.9723
5. System objects for MIMO	81.9175	5.0209
6. System objects for MIMO & Channel	59.9528	6.8604
7. System objects for MIMO & Channel & Viterbi	58.8660	6.9871
8. System objects for all	48.9014	8.4109
9. Version 8 + MATLAB to C code generation (MEX)	31.4722	13.0688
10. Version 8 + MEX + Parallel computing (parfor)	18.3899	22.3657
11. Version 8 Simulink normal mode	84.5959	4.8620
12. Version 8 Simulink normal mode optimized	44.3369	9.2768
13. Version 8 Simulink rapid accelerator	40.5559	10.1416
14. Version 8 Simulink rapid accelerator optimized	22.2629	18.4748

图 9.39　执行时间和十四个版本算法的加速率

Algorithm

MATLAB script: zPDCCH_vE_test

```
MaxSNR=8;
MaxNumBits=1e6;

fprintf(1,'\nVersion 14: Version 8 Simulink rapid accelerator optimized\n\n');
tic;
for EbNo=1:MaxSNR
   fprintf(1,'Iteration number %d\r',EbNo);
   sim('zPDCCH_v8s_optimized','SimulationMode','rapid',
'RapidAcceleratorUpToDateCheck', 'off');
end
time_14=toc;
fprintf(1,'Version 14: Time to complete %d iterations = %6.4f (sec)\n',
MaxSNR, time_14);
```

9.8.4.4 并行计算

现在，我们将并行计算引入 Simulink 的最快加速器模式。我们调用 Simulink 模型的最快加速器目标代码和 parfor 循环。首先，我们验证调用 matlabpool 命令和访问多核。通过使用 parfor，我们可以将 for 循环分配到多个 MATLAB 线程中执行（在本例中为两个线程）。下面的程序展示 parfor 循环如何调用 Simulink 模型在优化的最快加速器模式运行。这一过程与第十个版本的 MATLAB 代码并行化类似。

Algorithm

MATLAB script: zPDCCH_vF_test

```
MaxSNR=8;
MaxNumBits=1e6;
fprintf(1,'\nVersion 15: Version 8 Simulink rapid accel. optimized + parfor\n\n');
tic;
parfor EbNo=1:MaxSNR
   fprintf(1,'Iteration number %d\r',EbNo);
   sim('zPDCCH_v8s_optimized','SimulationMode','rapid',
'RapidAcceleratorUpToDateCheck', 'off');
end
time_15=toc;
fprintf(1,'Version 15: Time to complete %d iterations = %6.4f (sec)\n',
MaxSNR, time_15);
```

其结果显示，parfor 循环调用 Simulink 模型在优化的最快加速器模式模式编译执行大幅度提高了仿真速度。我们的仿真模型在默认模式下耗时 85.29s。而第十五个版本的优化模型仿真仅耗时 12.31s（见图 9.40），7 倍于第十一版，33

倍于 MATLAB 代码基准版（见图 9.41）。

```
Version 15: Version 8 Simulink rapid accel. optimized + parfor

Iteration number 6
Iteration number 5
Iteration number 4
Iteration number 8

Iteration number 3
Iteration number 2
Iteration number 1
Iteration number 7

Version 15: Time to complete 8 iterations = 12.3180 (sec)
```

图 9.40　第十五个版本的算法：循环八次耗时

Versions of the Transceiver	Elapsed Time (sec)	Acceleration Ratio
1. Baseline	411.3030	1.0000
2. Vectorization	326.5071	1.2597
3. Vectorization along larger dimension	175.8478	2.3390
4. Vectorization + Preallocation	82.7194	4.9723
5. System objects for MIMO	81.9175	5.0209
6. System objects for MIMO & Channel	59.9528	6.8604
7. System objects for MIMO & Channel & Viterbi	58.8660	6.9871
8. System objects for all	48.9014	8.4109
9. Version 8 + MATLAB to C code generation (MEX)	31.4722	13.0688
10. Version 8 + MEX + Parallel computing (parfor)	18.3899	22.3657
11. Version 8 Simulink normal mode	85.2923	4.8223
12. Version 8 Simulink normal mode optimized	44.3369	9.2768
13. Version 8 Simulink rapid accelerator	40.5559	10.1416
14. Version 8 Simulink rapid accelerator optimized	22.2629	18.4748
15. Version 8 Simulink rapid accel. optim. + parfor	12.3180	33.3903

图 9.41　执行时间和十五个版本算法的加速率

9.9　GPU 辅助运算

GPU 本来用于加速图像应用程序，不过现在被越来越多的应用于科学计算。MATLAB 有相关功能可以发挥 GPU 强大的运算能力。基于 CUDA（统一计算设备架构）的 NVIDIA GPU 计算可以参与 MATLAB 算法提速。FFT、反快速傅里叶变换（IFFT），和线性代数运算为代表的超过 100 个内建 MATLAB 函数可以通过 GPUArray 类型的输入变量声明调用 GPU 参与运算。并行计算工具包支持这种特殊 MATLAB 数组类型。这些 GPU 函数的操作根据它们传递数据的类型声明而各不相同。与此类似，如神经网络工具箱、通信系统工具箱、信号处理工具箱，和相阵列系统工具箱也支持 GPU 加速算法。

根据经验，一个计算密度高且并行度高的程序可能适合 GPU 运算。这可以解释为如下两条准则：第一，运行于 GPU 的程序会比运行中在 CPU 和 GPU 不断传输等量数据的程序耗时长的多。第二、当所有核都处于繁忙状态，需要使用

GPU 进行并行处理时，GPU 辅助运算才能发挥最大优势。大数组向量化 MATLAB 计算和 GPU 支持的工具箱函数满足这一类条件。

当访问 GPU 时，我们可以借助其力量动态提升 MATLAB 算法的仿真速度，特别是数据量非常大的时候。针对 GPU 的算法优化特别适合移动通信系统，因为移动通信系统往往有吞吐大数据量和多用户相互独立又重复的操作的特点。

9.9.1 在 MATLAB 中启动 GPU 功能

使用 GPU 运行如下 MATLAB 例子需要使用并行计算工具箱和通信系统工具箱。下面的命令可以验证许可证是否包含这两个工具箱：

Algorithm

```
license('test','distrib_computing_toolbox');
license('test','communication_toolbox');
```

假如 MATLAB 显示结果都为 1，则表示两个工具箱的许可证可用。为了验证 GPU 是否正确安装并可用于 MATLAB，我们可以输入如下命令验证：

Algorithm

```
parallel.gpu.GPUDevice.isAvailable
```

假如返回值为 1，则表示 MATLAB 可以使用 GPU。

9.9.2 GPU 优化系统对象

通信系统工具箱有很多特定的算法支持 GPU 处理。并行计算工具箱可以在 GPU 上直接执行很多通信算法。下面为通信系统工具箱中支持 GPU 优化的系统对象：

- *comm.gpu.AWGNChannel*
- *comm.gpu.BlockDeinterleaver*
- *comm.gpu.BlockInterleaver*
- *comm.gpu.ConvolutionalDeinterleaver*
- *comm.gpu.ConvolutionalEncoder*
- *comm.gpu.ConvolutionalInterleaver*
- *comm.gpu.LDPCDecoder*
- *comm.gpu.PSKDemodulator*
- *comm.gpu.PSKModulator*
- *comm.gpu.TurboDecoder*
- *comm.gpu.ViterbiDecoder.*

我们看到，并不是所有的系统对象都支持 GPU 优化。表中所列都是很多通信系统中高计算密度的算法。它们都有一个符号：.gpu 添加在对象名之中。通

过对代码进行些许改变，MATLAB 仿真可在 GPU 辅助的情况下得到速度提升。

9.9.3 使用单一 GPU 系统对象

下面的 MATLAB 函数为 PDCCH 算法第八版的第一个 GPU 优化版。MATLAB 代码的改变仅仅是用 comm. gpu. ViterbiDecoder 系统对象替换了 comm. ViterbiDecoder。

Algorithm

MATLAB function: zPDCCH_vG

```
function [ber, bits]=zPDCCH_vG(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2048;
M=4; k=log2(M); codeRate=1/3;
snr = EbNo + 10*log10(k) + 10*log10(codeRate);
trellis=poly2trellis(7, [133 171 165]);
L=FRM+24;C=6; Index=[L+1:(3*L/2) (L/2+1):L];
%% Initializations
persistent Modulator AWGN DeModulator BitError ConvEncoder1 ConvEncoder2 Viterbi
CRCGen CRCDet
if isempty(Modulator)
   Modulator     = comm.QPSKModulator('BitInput',true);
   AWGN          = comm.AWGNChannel('NoiseMethod', 'Variance', 'VarianceSource',
'Input port');
   DeModulator =  comm.QPSKDemodulator('BitOutput',true);
   BitError       = comm.ErrorRate;
   ConvEncoder1=comm.ConvolutionalEncoder('TrellisStructure', trellis,
'FinalStateOutputPort', true, ...
      'TerminationMethod','Truncated');
   ConvEncoder2 =  comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'InitialStateInputPort', true,...
      'TrellisStructure', trellis);
   Viterbi=comm.gpu.ViterbiDecoder('TrellisStructure', trellis,
'InputFormat','Hard','TerminationMethod','Truncated');
   CRCGen = comm.CRCGenerator('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
   CRCDet = comm.CRCDetector   ('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
end
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
results=zeros(3,1);
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u           = randi([0 1], FRM,1);                         % Generate bit payload
   u1          = step(CRCGen, u);                              % CRC insertion
   u2          = u1((end-C+1):end);                           % Tail-biting convolutional coding
   [~, state]  = step(ConvEncoder1, u2);
   u3          = step(ConvEncoder2, u1,state);
   u4          = fcn_RateMatcher(u3, L, codeRate); % Rate matching
```

```
u5          = fcn_Scrambler(u4, nS);                 % Scrambling
u6          = step(Modulator, u5);                  % Modulation
u7          = TransmitDiversityEncoderS(u6);         % MIMO Alamouti encoder
 % Channel
[u8, h8]    = MIMOFadingChanS(u7);                   % MIMO fading channel
noise_var = real(var(u8(:)))/(10.^(0.1*snr));
u9          = step(AWGN, u8, noise_var);             % AWGN
% Receiver
uA          =  TransmitDiversityCombinerS(u9, h8);% MIMO Alamouti combiner
uB          = step(DeModulator, uA);                % Demodulation
uC          = fcn_Descrambler(uB, nS);               % Descrambling
uD          = fcn_RateDematcher(uC, L);              % Rate de-matching
uE          = [uD;uD];                               % Tail-biting
uF          = step(Viterbi, uE);                     % Viterbi decoding
uG          = uF(Index);
y           = step(CRCDet, uG );                      % CRC detection
results     = step(BitError, u, y);                  % Update number of bit errors
numErrs     = results(2);
numBits     = results(3);
nS          = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = results(1); bits= results(3);
reset(BitError);
```

通过运行下面的 MATLAB 脚本，调用第一个 GPU 优化版算法，并对比性能提升情况，我们可以发现使用 GPU 对算法瓶颈（Viterbi 译码器）的优化作用。注意其他的组件都工作在 CPU（见图 9.42 和图 9.43）。

```
Version 16: Version 8 + Viterbi decoder on GPU

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 16: Time to complete 8 iterations = 26.5780 (sec)
```

图 9.42 第十六个版本的算法：循环八次耗时

```
------------------------------------------------------------------------------------------------------
Versions of the Transceiver                              | Elapsed Time (sec)| Acceleration Ratio
 1. Baseline                                             |         411.3030 |      1.0000
 2. Vectorization                                        |         326.5071 |      1.2597
 3. Vectorization along larger dimension                 |         175.8478 |      2.3390
 4. Vectorization + Preallocation                        |          82.7194 |      4.9723
 5. System objects for MIMO                              |          81.9175 |      5.0209
 6. System objects for MIMO & Channel                    |          59.9528 |      6.8604
 7. System objects for MIMO & Channel & Viterbi          |          58.8660 |      6.9871
 8. System objects for all                               |          48.9014 |      8.4109
 9. Version 8 + MATLAB to C code generation (MEX)        |          31.4722 |     13.0688
10. Version 8 + MEX + Parallel computing (parfor)        |          18.3899 |     22.3657
11. Version 8 Simulink normal mode                       |          85.2923 |      4.8223
12. Version 8 Simulink normal mode optimized             |          44.3369 |      9.2768
13. Version 8 Simulink rapid accelerator                 |          40.5559 |     10.1416
14. Version 8 Simulink rapid accelerator optimized       |          22.2629 |     18.4748
15. Version 8 Simulink rapid accel. optim. + rarfor      |          12.3180 |     33.3904
16. Version 8 + Viterbi decoder on GPU                   |          26.5780 |     15.4753
------------------------------------------------------------------------------------------------------
```

图 9.43　执行时间和十六个版本算法的加速率

Algorithm

MATLAB function

```
fprintf(1,'\nVersion 16: Version 8 + Viterbi decoder on GPU\n\n');
tic;
for snr = 1:MaxSNR
ber= zPDCCH_vG(snr, MaxNumBits, MaxNumBits);
end
time_16=toc;
fprintf(1,'Version 16: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_16);
```

9.9.4 GPU 参与并行计算

在这一版本的算法中，我们同时使用 GPU 和并行计算。为了并行化算法，我们使用并行处理工具箱中的 spmd 函数。这个函数实现了一个“单程序，多数据”架构，在一系列 MATLAB 线程中同时执行 MATLAB 程序。spmd 语句的一般格式如下：

Algorithm

```
spmd;
<MATLAB statements>
end;
```

为了能并行执行此语句，我们必须首先用前面介绍的 matlabpool 函数打开一个 MATLAB 线程池。在 spmd 体中，每个 MATLAB 线程有各不相同的标识，这些标识由 labindex 变量和表示并行执行区块所有线程数的变量 numlabs 确定。

下面的函数为我们 PDCCH 算法的最终版本。他使用 spmd 函数并行化循环内运算，并针对 Viterbi 译码器使用 GPU 优化系统对象。

Algorithm

MATLAB function

```
function [ber, bits]=zPDCCH_vH(EbNov, maxNumErrs, maxNumBits)
%% Constants
wkrs = 2;
spmd(wkrs)
FRM=2048;
M=4; k=log2(M); codeRate=1/3;
snrv = EbNov + 10*log10(k) + 10*log10(codeRate);
bits=zeros(size(EbNov));errs=bits;
trellis=poly2trellis(7, [133 171 165]);
L=FRM+24;C=6; Index=[L+1:(3*L/2) (L/2+1):L];
s = RandStream.create('mrg32k3a', 'NumStreams', wkrs, 'CellOutput', true, 'Seed', 1);
RandStream.setGlobalStream(s{labindex});
Modulator     = comm.QPSKModulator('BitInput',true);
AWGN          = comm.AWGNChannel('NoiseMethod', 'Variance', 'VarianceSource',
'Input port');
DeModulator =  comm.QPSKDemodulator('BitOutput',true);
BitError      = comm.ErrorRate;
ConvEncoder1=comm.ConvolutionalEncoder('TrellisStructure', trellis,
'FinalStateOutputPort', true, 'TerminationMethod','Truncated');
ConvEncoder2 =  comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'InitialStateInputPort', true,'TrellisStructure', trellis);
Viterbi=comm.gpu.ViterbiDecoder('TrellisStructure', trellis,
'InputFormat','Hard','TerminationMethod','Truncated');
CRCGen = comm.CRCGenerator('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
CRCDet = comm.CRCDetector   ('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
%end
for n=1:numel(snrv),
  %% Processing loop modeling transmitter, channel model and receiver
  numErrs = 0; numBits = 0; nS=0;
  results=zeros(3,1);
  while ((numErrs < maxNumErrs/numlabs) && (numBits < maxNumBits/numlabs))
    % Transmitter
    u           = randi([0 1], FRM,1);                    % Generate bit payload
    u1          = step(CRCGen, u);                         % CRC insertion
    u2          = u1((end-C+1):end);                       % Tail-biting convolutional coding
    [~, state]  = step(ConvEncoder1, u2);
    u3          = step(ConvEncoder2, u1,state);
    u4          = fcn_RateMatcher(u3, L, codeRate);  % Rate matching
    u5          = fcn_Scrambler(u4, nS);                   % Scrambling
    u8          = step(Modulator, u5);                     % Modulation
    u7          = TransmitDiversityEncoderS(u8);           % MIMO Alamouti encoder
```

```
        % Channel
        [u8, h8]    = MIMOFadingChanS(u7);                    % MIMO fading channel
        noise_var = real(var(u8(:)))/(10.^(0.1*snrv(n)));
        u9          = step(AWGN, u8, noise_var);              % AWGN
        % Receiver
        uA          =  TransmitDiversityCombinerS(u9, h8);% MIMO Alamouti combiner
        uB          = step(DeModulator, uA);                  % Demodulation
        uC          = fcn_Descrambler(uB, nS);                % Descrambling
        uD          = fcn_RateDematcher(uC, L);               % Rate de-matching
        uE          = [uD;uD];                                % Tail-biting
        uF          = step(Viterbi, uE);                      % Viterbi decoding
        uG          = uF(Index);
        y           = step(CRCDet, uG );                      % CRC detection
        results     = step(BitError, u, y);                   % Update number of bit errors
        numErrs     = results(2);
        numBits     = results(3);
        nS          = nS + 2; nS = mod(nS, 20);
    end
    %% Clean up & collect results
    bits(n)= results(3);
    errs(n) = results(2);
    reset(BitError);
end
end
totbits = zeros(1, numel(EbNov));
toterrs = zeros(1, numel(EbNov));
for n=1:wkrs,
    totbits = totbits + bits{n};
    toterrs = toterrs + errs{n};
end
ber = toterrs./totbits;
```

函数内遍历所有的 Eb/N0 输入值并将其全部存贮在一个向量内，而并不是一个一个值输入。不仅如此，在 spmd 函数体内我们在独立线程上进行相同的运算。为了得到有效的 BER 值，我们必须确保随机数生成器产生随机比特且与作为 AWGN 噪声的值无关。这个工作由下面两行代码进行，他们由不同线程生成不同的随机数。

Algorithm

```
s = RandStream.create('mrg32k3a', 'NumStreams', wkrs, 'CellOutput', true, 'Seed', 1);
RandStream.setGlobalStream(s{labindex});
```

我们同时根据线程数分配最大误码数和最大比特数，如每个线程处理同样多的比特。在 spmd 的结尾，每个线程独立计算得到包含不同值的变量。在 for 循环中，在 spmd 之后，我们将所有线程计算得到的比特和 BER 值相加得到

总 BER。

下面的调用函数执行这一版本的算法评估 GPU 和并行计算对仿真速度的提升。结果显示其在生成相同数值结果的情况下耗时更少（见图 9.44 和图 9.45）。

```
Version 17: Version 8 + Viterbi decoder on GPU + spmd

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8
Version 17: Time to complete 8 iterations = 15.2433 (sec)
```

图 9.44 第十七个版本的算法：循环八次耗时

Versions of the Transceiver	Elapsed Time (sec)	Acceleration Ratio
1. Baseline	411.3030	1.0000
2. Vectorization	326.5071	1.2597
3. Vectorization along larger dimension	175.8478	2.3390
4. Vectorization + Preallocation	82.7194	4.9723
5. System objects for MIMO	81.9175	5.0209
6. System objects for MIMO & Channel	59.9528	6.8604
7. System objects for MIMO & Channel & Viterbi	58.8660	6.9871
8. System objects for all	48.9014	8.4109
9. Version 8 + MATLAB to C code generation (MEX)	31.4722	13.0688
10. Version 8 + MEX + Parallel computing (parfor)	18.3899	22.3657
11. Version 8 Simulink normal mode	85.2923	4.8223
12. Version 8 Simulink normal mode optimized	44.3369	9.2768
13. Version 8 Simulink rapid accelerator	40.5559	10.1416
14. Version 8 Simulink rapid accelerator optimized	22.2629	18.4748
15. Version 8 Simulink rapid accel. optim. + rarfor	12.3180	33.3904
16. Version 8 + Viterbi decoder on GPU	26.5780	15.4753
17. Version 8 + Viterbi decoder on GPU + spmd	15.2433	26.9825

图 9.45 执行时间和十七个版本算法的加速率

Algorithm

MATLAB function

```
fprintf(1,'\nVersion 17: Version 8 + Viterbi decoder on GPU + spmd\n\n');
tic;
ber= zPDCCH_vH(1:snr, MaxNumBits, MaxNumBits);
time_17=toc;
fprintf(1,'Version 17: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_17);
```

9.10 实例研究：在 GPU 上进行 Turbo 编码

在本节中我们进行的实例研究为如何通过 GPU 对 MATLAB 编写的 Turbo 编码应用程序进行提速。Turbo 编码是 LTE 标准的关键组件。因为这个算法本身基于迭代循环，故 Turbo 译码器的计算密度很高，适合用 GPU 加速。在 Turbo 译码中使用 GPU 优化系统对象可以提高 BER 仿真速度。

我们首先重复前面几节的工作，只是将 Viterbi 译码器替换为 Turbo 译码器。然后我们会解决几个 MATLAB 中使用 GPU 处理会遇到的意想不到的问题。一个主要的问题是 GPU 和 CPU 之间过量的数据传输。这一个 GPU 处理中常见的问题且会降低仿真速度。最后，我们会讲解如何增大由 GPU 负责处理的数据大小从而进一步加速仿真。

9.10.1 基于 CPU 处理基准算法

我们用第 8 章中的 Turbo 码算法作为基准。下面的函数用 CPU 处理执行此算法。输入数据大小为每帧 2432bit。LTE 标准所定义了 Turbo 编码使用的网格结构和交织器。在发射端中，Turbo 编码器在 QPSK 调制器之后。为了简化代码，信道建模仅为 AWGN 信道模型。在接收端我们在 Turbo 译码器之后使用软判决解调。

Algorithm

MATLAB function: zTurboExample_gpu0

```
function [ber, bits]=zTurboExample_gpu0(EbNo, maxNumErrs, maxNumBits)
FRM=2432;
Indices = lteIntrlvrIndices(FRM);
M=4;k=log2(M);
R= FRM/(3* FRM + 4*3);
snr = EbNo + 10*log10(k) + 10*log10(R);
noiseVar = 10.^(-snr/10);
numIter=6; trellis =  poly2trellis(4, [13 15], 13);
persistent hTEnc Modulator AWGN DeModulator hTDec hBER
if isempty(Modulator)
   hTEnc = comm.TurboEncoder('TrellisStructure',trellis , 'InterleaverIndices', Indices);
   Modulator      = comm.PSKModulator(4, ...
      'BitInput', true, 'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
      'CustomSymbolMapping', [0 2 3 1]);
   AWGN            = comm.AWGNChannel('NoiseMethod', 'Variance', 'VarianceSource',
'Input port');
   DeModulator =  comm.PSKDemodulator(...
      'ModulationOrder', 4, ...
      'BitOutput', true, ...
```

```
    'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
    'CustomSymbolMapping', [0 2 3 1],...
    'DecisionMethod', 'Approximate log-likelihood ratio', ...
    'VarianceSource', 'Input port');
  % Turbo Decoder
  hTDec  = comm.TurboDecoder('TrellisStructure', trellis,'InterleaverIndices', Indices,
'NumIterations', numIter);
  % BER measurement
  hBER = comm.ErrorRate;
end
%% Processing loop
Measures = zeros(3,1); %initialize BER output
while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
  data = randi([0 1], FRM, 1);
  % Encode random data bits
  yEnc = step(hTEnc, data);
  % Add noise to real bipolar data
  modout = step(Modulator, yEnc);
  rData = step(AWGN, modout,noiseVar );
  % Convert to log-likelihood ratios for decoding
  llrData = step( DeModulator, rData, noiseVar);
  % Turbo Decode
  decData = step(hTDec, -llrData);
  % Calculate errors
  Measures = step(hBER, data, decData);
end
bits = Measures(3);
ber= Measures(1);
reset(hBER);
```

我们可以使用分析器查找这个算法的瓶颈，如以下命令所示。结果清楚的显示，算法处理全部 1 百万比特数据共耗时 26.503s，而 Turbo 译码器耗时 24.485s，所以 Turbo 译码器是算法的主要瓶颈（见图 9.46）。

Algorithm

MATLAB script

```
EbNo=0; maxNumErrs=1e6;maxNumBits=1e6;
profile on;
ber= zTurboExample_gpu0(EbNo, maxNumErrs, maxNumBits);
profile viewer;
```

为了建立一个 Turbo 码执行时间的衡量标尺，我们运行下面的 MATLAB 脚本。Eb/N0 值以 0.2dB 补偿从 0 到 1.2dB 取值，共七次循环。脚本处理 1 百万比特数据共需 311s（见图 9.47）。

Profile Summary

Generated 16-Jan-2013 19:03:39 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
zTurboExample_gpu0	1	26.503 s	0.748 s	
TurboDecoder>TurboDecoder.stepImpl	412	24.485 s	24.426 s	
TurboEncoder>TurboEncoder.stepImpl	412	0.440 s	0.331 s	
AWGNChannel>AWGNChannel.stepImpl	412	0.339 s	0.282 s	

图 9.46　基准 Turbo 算法的分析结果

```
Version 1: Everything on CPU

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Version 1: Time to complete 7 iterations = 311.2196 (sec)
-------------------------------------------------------------------------------------------
Versions of the Transceiver                          | Elapsed Time (sec)| Acceleration Ratio
 1. Everything on CPU                                |          311.2196 |       1.0000
-------------------------------------------------------------------------------------------
```

图 9.47　执行时间和基准 Turbo 算法的加速率

Algorithm

MATLAB script

```
MaxSNR=7;
Snrs=0:0.2:1.2;
MaxNumBits=1e6;
N=1;
fprintf(1,'\nVersion 1: Everything on CPU\n\n')
tic;
for idx = 1:MaxSNR
   fprintf(1,'Iteration number %d\r',idx);
   EbNo=Snrs(idx);
   ber= zTurboExample_gpu0(EbNo, MaxNumBits, MaxNumBits);
end
time_CPU=toc;
fprintf(1,'Version 1: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR, time_CPU);
Report_Timing_Results(N,time_CPU,time_CPU,'Everything on CPU');
```

9.10.2 基于 GPU 处理 Turbo 译码器

在第二个版本的 Turbo 编码例子中，我们处理上一节所述的瓶颈，用 GPU 在 CPU 执行算法之余执行 Turbo 译码器。用 comm. gpu. TurboDecoder 系统对象替换 comm. TurboDecoder 就可解决问题。下面的函数为基准算法（zTurboExample _ gpu0）与第二版（zTurboExample _ gpu1）有区别的地方。

MATLAB function

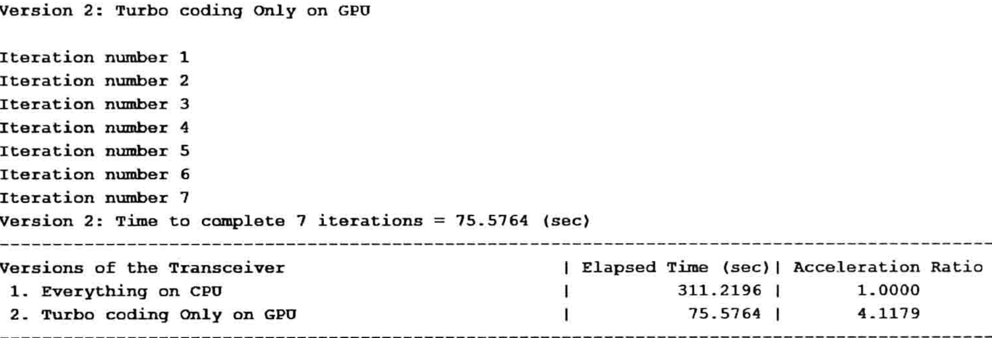
function [ber, bits] = zTurboExample_gpu0(EbNo, maxNumErrs, maxNumBits)	function [ber, bits] = zTurboExample_gpu1(EbNo, maxNumErrs, maxNumBits)
. hTDec = comm.TurboDecoder ('TrellisStructure', trellis,'InterleaverIndices', Indices, 'NumIterations', numIter); . . end	. hTDec = comm.gpu.TurboDecoder ('TrellisStructure', trellis,'InterleaverIndices', Indices, 'NumIterations', numIter); . . end

我们运行以下的 MATLAB 脚本，可以评估 GPU 是否对 Turbo 译码器提速有帮助。结果显示七个循环处理 1 百万比特数据耗时 75s：速度提升了 4 倍（见图 9.48）。

Algorithm

MATLAB script

```
MaxSNR=7;
Snrs=0:0.2:1.2;
MaxNumBits=1e6;
N=2;
fprintf(1,'\nVersion 2: Turbo coding Only on GPU\n\n');
tic;
for idx = 1:MaxSNR
   fprintf(1,'Iteration number %d\r',idx);
   EbNo=Snrs(idx);
   ber= zTurboExample_gpu1(EbNo, MaxNumBits, MaxNumBits);
end
time_GPU1=toc;
fprintf(1,'Version 2: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR,
time_GPU1);
Report_Timing_Results(N,time_CPU,time_GPU1,'Turbo coding Only on GPU');
```



```
Version 2: Turbo coding Only on GPU

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Version 2: Time to complete 7 iterations = 75.5764 (sec)
-----------------------------------------------------------------------------------------------
Versions of the Transceiver                          | Elapsed Time (sec)| Acceleration Ratio
 1. Everything on CPU                                |          311.2196 |        1.0000
 2. Turbo coding Only on GPU                         |           75.5764 |        4.1179
-----------------------------------------------------------------------------------------------
```

图 9.48　执行时间和第二版 Turbo 算法的加速率

9.10.3　基于 GPU 处理多个系统对象

在第三个版本的 Turbo 编码例子中，我们除了使用 Turbo 译码器之外，另添加多个 GPU 优化的系统对象。它们包括 comm. gpu. PSKModulator、comm. gpu. AWGNChannel、comm. gpu. PSKDemodulator，和 comm. gpu. TurboDecoder。即算法的一部分在 CPU 运行，一部分在 GPU 运行。GPU 和 CPU 之间需要进行通信。gpuArray 函数负责从 CPU 向 GPU 通信，而 gather 函数负责从 GPU 向 CPU 通信。下面的 MATLAB 函数实现这一算法：

Algorithm

MATLAB function

```
function [ber, bits]=zTurboExample_gpu2(EbNo, maxNumErrs, maxNumBits)
FRM=2432;
Indices = lteIntrlvrIndices(FRM);
M=4;k=log2(M);
R= FRM/(3* FRM + 4*3);
snr = EbNo + 10*log10(k) + 10*log10(R);
noiseVar = 10.^(-snr/10);
numIter=6; trellis =  poly2trellis(4, [13 15], 13);
persistent hTEnc Modulator AWGN DeModulator hTDec hBER
if isempty(Modulator)
   hTEnc = comm.TurboEncoder('TrellisStructure',trellis , 'InterleaverIndices', Indices);
   Modulator     = comm.gpu.PSKModulator(4, ...
      'BitInput', true, 'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
      'CustomSymbolMapping', [0 2 3 1]);
```

```
  AWGN          =comm.gpu.AWGNChannel('NoiseMethod', 'Variance', 'VarianceSource',
'Input port');
  DeModulator =  comm.gpu.PSKDemodulator(...
     'ModulationOrder', 4, ...
     'BitOutput', true, ...
     'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
     'CustomSymbolMapping', [0 2 3 1],...
     'DecisionMethod', 'Approximate log-likelihood ratio', ...
     'VarianceSource', 'Input port');
  % Turbo Decoder
  hTDec  = comm.gpu.TurboDecoder('TrellisStructure', trellis,'InterleaverIndices', Indices,
'NumIterations', numIter);
  % BER measurement
  hBER = comm.ErrorRate;
end
%% Processing loop
Measures = zeros(3,1); %initialize BER output
while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
  data = randi([0 1], numFrames*FRM, 1);
  % Encode random data bits
  yEnc = gpuArray(step(hTEnc, data));
  % Modulate the signal - send to GPU
  modout = step(Modulator, yEnc);
  % Add noise to data
  rData = step(AWGN, modout,noiseVar );
  % Convert to log-likelihood ratios for decoding
  llrData = step( DeModulator, rData, noiseVar);
  % Turbo Decode
  decData = step(hTDec, -llrData);
  % Calculate errors
  Measures = step(hBER, data, gather(decData));
end
bits = Measures(3);
ber= Measures(1);
reset(hBER);
end
```

我们期待通过使用更多的 GPU 优化系统对象得到更快的仿真速度。但是，实际上 GPU 到 CPU 间的通信开销可能会抵消使用系统对象带来的速度提升。下面的 MATLAB 脚本运行结果说明了这一点。第三个版本的耗时要比第二个版本略长，七次循环共耗时 86s。注意到在 GPU 运行始终要比在 CPU 要快。第三个版本比基准 CPU 版本快 3. 5 倍，如图 9. 49 的结果所示。

```
Version 3:  Four GPU algorithms + Single-frame

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Version 3: Time to complete 7 iterations = 86.6899 (sec)
------------------------------------------------------------------------------------------
Versions of the Transceiver                    | Elapsed Time (sec)| Acceleration Ratio
 1. Everything on CPU                          |          311.2196 |       1.0000
 2. Turbo coding Only on GPU                   |           75.5764 |       4.1179
 3. Four GPU algorithms + Single-frame         |           86.6899 |       3.5900
------------------------------------------------------------------------------------------
```

图 9.49　执行时间和第三版 Turbo 算法的加速率

Algorithm

MATLAB function

```
MaxSNR=7;
Snrs=0:0.2:1.2;
MaxNumBits=1e6;
N=3;
fprintf(1,'\nVersion 3:  Four GPU algorithms + Single-frame\n\n');
tic;
for idx = 1:MaxSNR
   fprintf(1,'Iteration number %d\r',idx);
   EbNo=Snrs(idx);
   ber= zTurboExample_gpu2(EbNo, MaxNumBits, MaxNumBits);
end
time_GPU2=toc;
fprintf(1,'Version 3: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR,
time_GPU2);
Report_Timing_Results(N,time_CPU,time_GPU2,'Four GPU algorithms + Single-frame');
```

9.10.4　多帧和大数据长度

第四个版本的算法对 GPU 到 CPU 间过量通信造成的性能损失进行补偿。算法将输入数据与 GPU 并行处理运行的多帧进行并置。首先，GPU 处理适合使用大数据长度。其次，通过数据并置，使 Turbo 译码器之外的系统对象占据更多 GPU 缓存进行处理，从而减小了 gpuArray 和 gather 函数的负荷。下面的 MATLAB 函数为第四个版本的算法。

Algorithm

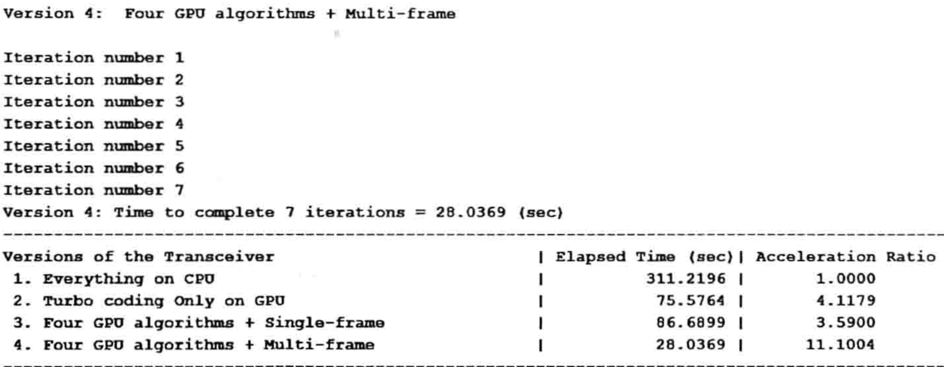
MATLAB function

```
function [ber, bits]=zTurboExample_gpu3(EbNo, maxNumErrs, maxNumBits)
FRM=2432;
Indices = lteIntrlvrIndices(FRM);
M=4;k=log2(M);
R= FRM/(3* FRM + 4*3);
snr = EbNo + 10*log10(k) + 10*log10(R);
noiseVar = 10.^(-snr/10);
numIter=6; trellis =  poly2trellis(4, [13 15], 13);
numFrames = 30;                          %Run 30 frames in parallel
persistent hTEnc Modulator AWGN DeModulator hTDec hBER
if isempty(Modulator)
  hTEnc = comm.TurboEncoder('TrellisStructure',trellis , 'InterleaverIndices', Indices);
  Modulator     = comm.gpu.PSKModulator(4, ...
     'BitInput', true, 'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
                                            'CustomSymbolMapping', [0 2 3 1]);
  AWGN         =comm.gpu.AWGNChannel('NoiseMethod', 'Variance', 'VarianceSource',
'Input port');
  DeModulator =  comm.gpu.PSKDemodulator(...
     'ModulationOrder', 4, ...
     'BitOutput', true, ...
     'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
     'CustomSymbolMapping', [0 2 3 1],...
     'DecisionMethod', 'Approximate log-likelihood ratio', ...
     'VarianceSource', 'Input port');
  % Turbo Decoder with MultiFrame processing
  hTDec  = comm.gpu.TurboDecoder('TrellisStructure', trellis,'InterleaverIndices', Indices,
...
     'NumIterations', numIter, 'NumFrames', numFrames);
  % BER measurement
  hBER = comm.ErrorRate;
end
%% Processing loop
  Measures = zeros(3,1); %initialize BER output
  while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
     data = randi([0 1], numFrames*FRM, 1);
     % Encode random data bits
     yEnc = gpuArray(multiframeStep(hTEnc, data, numFrames));
     % Add noise to real bipolar data
     modout = step(Modulator, yEnc);
     rData = step(AWGN, modout,noiseVar );
     % Convert to log-likelihood ratios for decoding
     llrData = step( DeModulator, rData, noiseVar);
     % Turbo Decode
     decData = step(hTDec, -llrData);
```

```
        % Calculate errors
        Measures = step(hBER, data, gather(decData));
    end
bits = Measures(3);
ber= Measures(1);
reset(hBER);
end
function y = multiframeStep(h, x, nf)
    xr = reshape(x,[], nf);
    ytmp = step(h,xr(:,1));
    y = zeros(size(ytmp,1), nf, class(ytmp));
    y(:,1) = ytmp;
    for ii =2:nf,
        y(:,ii) = step(h,xr(:,ii));
    end
    y = reshape(y, [], 1);
end
```

第四版对第三版算法做了如下更改。使用一个叫 numFrames 的变量表示仿真中多少数据帧进行了并置。我们设定 numFrames 参数值为 30。这一参数在 GPU 并行化处理译码操作的过程中调用。函数同时调用 multiframeStep 表征多次进行 Turbo 译码操作和并置的结果。

下面的 MATLAB 脚本遍历 SNR 值并记录耗时。我们可以注意到算法上一点点的更改，使仿真速度得到巨大提升。第四个版本的算法耗时 28s，如图 9.50 所示。



```
Version 4:  Four GPU algorithms + Multi-frame

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Version 4: Time to complete 7 iterations = 28.0369 (sec)
```

Versions of the Transceiver	Elapsed Time (sec)	Acceleration Ratio
1. Everything on CPU	311.2196	1.0000
2. Turbo coding Only on GPU	75.5764	4.1179
3. Four GPU algorithms + Single-frame	86.6899	3.5900
4. Four GPU algorithms + Multi-frame	28.0369	11.1004

图 9.50　执行时间和第四版 Turbo 算法的加速率

Algorithm

MATLAB function

```
MaxSNR=7;
Snrs=0:0.2:1.2;
MaxNumBits=1e6;
N=4;
fprintf(1,'\nVersion 4:  Four GPU algorithms + Multi-frame\n\n');
tic;
for idx = 1:MaxSNR
   fprintf(1,'Iteration number %d\r',idx);
   EbNo=Snrs(idx);
   ber= zTurboExample_gpu3(EbNo, MaxNumBits, MaxNumBits);
end
time_GPU3=toc;
fprintf(1,'Version 3: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR,
time_GPU3);
Report_Timing_Results(N,time_CPU,time_GPU3,'Four GPU algorithms + Multi-frame');
```

9.10.5 使用单精度数据类型

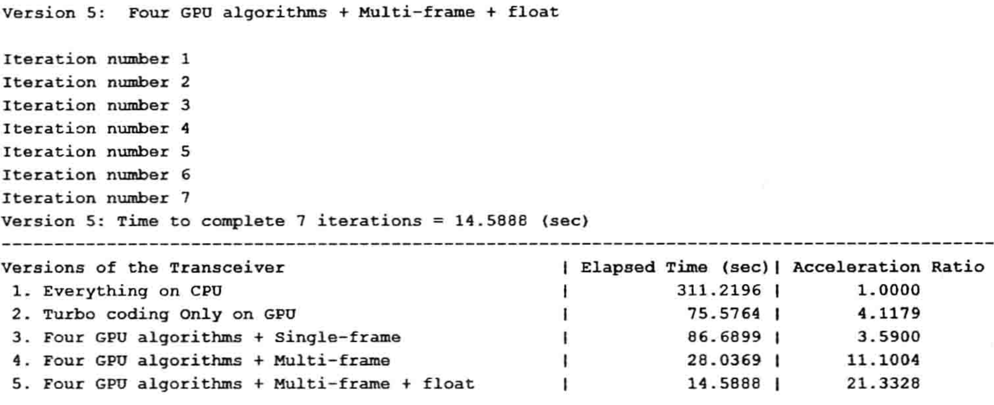
通过使用单精度浮点数据类型也可以提升仿真速度。GPU 运算对单精度数据类型进行了优化，所以我们需要确保第五版算法的所有变量都是单精度类型。调用 MATLAB 函数 single 可以将函数输出和变量更新为单精度类型。第五个版本的算法（zTurboExample _ gpu4）在 GPU 和 CPU 运算时都使用单精度浮点数，如下所示。

Algorithm

MATLAB function

```
function [ber, bits]=zTurboExample_gpu4(EbNo, maxNumErrs, maxNumBits)
FRM=2432;
Indices = single(lteIntrlvrIndices(FRM));
M=4;k=log2(M);
R= FRM/(3* FRM + 4*3);
snr = EbNo + 10*log10(k) + 10*log10(R);
noiseVar = single(10.^(-snr/10));
numIter=6; trellis =  poly2trellis(4, [13 15], 13);
numFrames = 30;                         %Run 30 frames in parallel
persistent hTEnc Modulator AWGN DeModulator hTDec hBER
if isempty(Modulator)
   hTEnc = comm.TurboEncoder('TrellisStructure',trellis , 'InterleaverIndices', Indices);
   Modulator     = comm.gpu.PSKModulator(4,'OutputDataType', 'single', ...
```

```
        'BitInput', true, 'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
                                              'CustomSymbolMapping', [0 2 3 1]);
    AWGN         =comm.gpu.AWGNChannel('NoiseMethod', 'Variance', 'VarianceSource',
'Input port');
    DeModulator =  comm.gpu.PSKDemodulator(...
        'ModulationOrder', 4, ...
        'BitOutput', true, ...
        'PhaseOffset', pi/4, 'SymbolMapping', 'Custom', ...
        'CustomSymbolMapping', [0 2 3 1],...
        'DecisionMethod', 'Approximate log-likelihood ratio', ...
        'VarianceSource', 'Input port');
    % Turbo Decoder with MultiFrame processing
    hTDec  = comm.gpu.TurboDecoder('TrellisStructure', trellis,'InterleaverIndices', Indices,
...
        'NumIterations', numIter, 'NumFrames', numFrames);
    % BER measurement
    hBER = comm.ErrorRate;
end
%% Processing loop
    Measures = zeros(3,1); %initialize BER output
    while (( Measures(2)< maxNumErrs) && (Measures(3) < maxNumBits))
        data = randi([0 1], numFrames*FRM, 1);
        % Encode random data bits
        yEnc = gpuArray(multiframeStep(hTEnc, data, numFrames));
        % Add noise to real bipolar data
        modout = step(Modulator, yEnc);
        rData = step(AWGN, modout,noiseVar );
        % Convert to log-likelihood ratios for decoding
        llrData = step( DeModulator, rData, noiseVar);
        % Turbo Decode
        decData = step(hTDec, -llrData);
        % Calculate errors
        Measures = step(hBER, data, gather(decData));
    end
bits = Measures(3);
ber= Measures(1);
reset(hBER);
end
function y = multiframeStep(h, x, nf)
    xr = reshape(x,[], nf);
    ytmp = step(h,xr(:,1));
    y = zeros(size(ytmp,1), nf, class(ytmp));
    y(:,1) = ytmp;
    for ii =2:nf,
        y(:,ii) = step(h,xr(:,ii));
    end
    y = reshape(y, [], 1);
end
```



```
Version 5:  Four GPU algorithms + Multi-frame + float

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Version 5: Time to complete 7 iterations = 14.5888 (sec)
--------------------------------------------------------------------------------------------
Versions of the Transceiver                          | Elapsed Time (sec)| Acceleration Ratio
 1. Everything on CPU                                |          311.2196 |     1.0000
 2. Turbo coding Only on GPU                         |           75.5764 |     4.1179
 3. Four GPU algorithms + Single-frame               |           86.6899 |     3.5900
 4. Four GPU algorithms + Multi-frame                |           28.0369 |    11.1004
 5. Four GPU algorithms + Multi-frame + float        |           14.5888 |    21.3328
--------------------------------------------------------------------------------------------
```

图 9.51 执行时间和第五版 Turbo 算法的加速率

我们可以通过下面的 MATLAB 脚本运行结果看到优化效果。第五版算法仅用时 14s，为第四版算法的两倍（双精度版）。总的来说，通过使用全部加速技术，优化算法和比 CPU 基准版本相比提速了 21 倍。结果总结如图 9.51 所示。

Algorithm

MATLAB function

```
MaxSNR=7;
Snrs=0:0.2:1.2;
MaxNumBits=1e6;
N=5;
fprintf(1,'\nVersion 5:  Four GPU algorithms + Multi-frame + float\n\n');
tic;
for idx = 1:MaxSNR
   fprintf(1,'Iteration number %d\r',idx);
   EbNo=Snrs(idx);
   ber= zTurboExample_gpu4(EbNo, MaxNumBits, MaxNumBits);
end
time_GPU4=toc;
fprintf(1,'Version 4: Time to complete %d iterations = %6.4f (sec)\n', MaxSNR,
time_GPU4);
Report_Timing_Results(N,time_CPU,time_GPU4,'Four GPU algorithms + Multi-
frame + float');
```

9.11 本章小结

在本章中，我们介绍了多种提高 MATLAB 和 Simulink 仿真速度的加速技术。我们进行了多种优化以提高仿真速度 LTE 控制信道算法和 Turbo 编码算法。首先

我们设定一个基准，随后陆续通过分析和代码更新介绍了如下优化方法：

1）更好的 MATLAB 串行编程技术（向量化、预分配）；

2）系统对象；

3）MATLAB - C 代码转换（MEX）；

4）并行计算（parfor，spmd）；

5）GPU 优化的系统模型；

6）Simulink 的最快加速器仿真模式。我们通过相近的 MATLAB 和 Simulink 实例阐明了通过几种或更多的仿真加速技术相结合可以使加速程度取得更多提升。

10 基于 C/C + +代码的原型构建

在前面各章，我们基于 MATLAB 和 Simulink 模型在 MATLAB 环境下对 LTEPHY层进行了仿真。在通信系统设计的一些阶段，我们需要开发一个软件组件可以脱离 MATLAB 环境直接进行仿真。例如，我们需要开发一个 C/C + +软件和对接外部仿真环境接口。这里有两种办法可以将 MATLAB 建模和仿真的结果导入到外部 C/C + +变成环境：我们可以手动的将 MTALB 开发的算法转换为 C/C + +程序，或者使用 MATLAB 中自动 C 代码生成功能。

通过使用 MATLAB 代码转换器，我们可以将 MATLAB 代码转换成独立的 C 和 C + +代码。生成的源代码有可移植性和可读性。MATLAB 代码转换器支持 MATLAB 语言的子集，包括程序控制结构、函数，和矩阵操作。代码转换器可以生成 MATLAB 可执行文件（MEX）函数用于加速 MATLAB 高计算密度仿真和验证。它还可以生成 C/C + +源代码，集成外部的 C 代码，建立可执行原型，或直接通过 C/C + +编译器在数字信号处理器（DSP）或通用 CPU 上执行。

在本章中，我们将讲解用 MATLAB 代码转换器从 MATLAB 代码生成独立的 C 和 C + +代码这一过程。我们首先说明其应用范围、技术动机，和 C/C + +代码生成要求。随后考察代码生成的两种方法：

1）在 MATLAB 命令行调用代码生成函数；

2）使用 MATLAB 代码转换器程序。

我们随后关注 MATLAB 这一功能支持范围，重点在各个系统工具箱的支持和各种数据类型的支持，包括定点类型数，和 MATLAB 程序使用的变长类型数。最后，我们对 MATLAB 算法转换代码集成外部 C/C + +测试平台的工作流程做一个说明。

10.1 应用范围

在我们讨论从 MATLAB 生成 C 代码之前，让我们首先解释为什么工程师需要将 MATLAB 代码转换为 C：

1）集成：我们希望将我们的 MATLAB 算法集成到外部 C 项目或软件中，如用户的仿真器、源代码或库文件。

2）原型构建：我们需要建立一个独立的原型或可执行文件以用于测试或用于方案验证。

3）加速：我们需要在 MATLAB 中执行用 C 代码编写的 MEX 文件。这一应用主要针对加速高计算密度的算法。

4）实现：我们需要在一个较大的系统设计中用 C 代码编写的嵌入式处理实现功能。

10.2 动机

通过 MATLAB—C 的算法自动转换，我们可以节省重写底层 C 代码和调试的时间，从而将更多时间用于在 MATLAB 环境中进行顶层开发和调试算法。在我们每次更新 MATLAB 代码的过程中，可以自动生成 MEX 文件。我们可以在 MATLAB 环境中用这个 MEX 文件验证代码是否正确编译。MEX 文件也可在大多数情况用于代码提速。该功能还可以自动生成源代码、可执行文件，或库文件。因此，我们在 MATLAB 中维护我们的设计的同时就可以周期性得到更新的 C/C + +代码。MATLAB 简单的软件参考可以轻松实现程序更改或性能提升。如本章下面要讨论的，我们还可以使用自动化工具帮助评估 MATLAB 代码的可读性，以便于代码生成。这些工具可以指导我们如何成功地将 MATLAB 算法转换为 C 代码。

10.3 要求

为了从 MATLAB 算法生成 C/C + +代码，我们首先要安装 MATLAB 代码转换器并使用 C/C + +编译器。首先，我们对编译器进行设置。对大多数平台来说，MATLAB 集成了 MathWorks 的默认编译器。假如安装不包含这个默认编译器的话，我们必须安装一个 MATLAB 支持的 C/C + +编译器。MATLAB 文档中包含一个所支持编译器的列表[1]。设置安装好的编译器，需要在 MATLAB 命令行输入：

Algorithm

```
>> mex -setup
```

输出会显示安装好的编译器列表并可从中选择一个。注意编译器的选择至关重要，因为 MATLAB 代码编译的仿真速度与使用的编译器种类和编译器选项密切相关。

本书中仿真得到数值和时间结果与 MATLAB 安装平台、操作系统、C/C + +编译器或 GPU 的类型相关。本书中非 GPU 程序运行的笔记本电脑配置如下：

1）硬件：2.70GHz Intel Dual - Core i7 - 2620M CPU，8GB RAM

2）操作系统：64位Windows 7 Enterprise（SP1）

3）C/C++编译器：Microsoft Visual Studio 2010与Microsoft Windows SDK7.1

10.4 MATLAB代码的构思

为了将MATLAB代码转换为高效的C/C++代码，我们必须时刻考量如下MATLAB代码属性：

1）数据类型：C和C++是静态类型语言。MATLAB，从某种意义上说是动态数据类型语言。为了消除这两者之间的阻碍，MATLAB代码转换器为了正确生成代码，需要对每个变量有一个完整的类型分配。MATLAB代码转换器有各种途径在调用前确定变量类型。对每个变量而言，有三个属性必须确定：类（或数据类型）、长度（或阶数），以及复变性（是否是复变量）。如在下一章中会展示MATLAB在转换C/C++时如何轻松的定义这些属性。

2）数组长度：MATLAB中的变量长度（阶数）在仿真中可以是固定的或可变的。代码生成支持可变长度数组和矩阵。我们可以定义MATLAB函数的输入、输出，和本地变量以表示数据的长度实时可变。

3）内存分配：我们可以选择生成的代码使用动态或静态内存分配方式。使用动态内存分配的代码速度最快，不过内存占有率较高。因此，动态内存分配一般适用于既存的MATLAB代码不需要修改生成C代码的情况。动态内存分配也允许一些无法找到最大字段值的程序编译。静态分配减小生成代码的内存占有率，也因此适用于可用内存空间有限的应用程序，如嵌入式应用。

4）速度：对涉及实时信号处理的应用，算法上必须要求足够快的速度与数据接收速率相吻合。一种提升实时应用代码速度的办法是停用不必要的实时校验。实时校验包括确认数组边界完整性、响应性和避免生成无效数值结果——如除零运算——这类操作的额外代码。我们在验证算法的设计正确无误之后，可以停用这些校验以提高生成的C代码速度，例如，连续分配数组边界的情况。

10.5 如何创建代码

在本节中我们会回顾自动MATLAB—C代码转换的一般步骤。代码既可以用MATLAB代码转换器工程或在MATLAB命令栏输入codegen命令实现。

10.5.1 实例研究：频域均衡

我们首先从一个简单的例子开始。这个例子实现频域均衡算法。在这个算法中，输出y为接收输入信号u采样和信道增益信号系数采样的乘积。

Algorithm

MATLAB function: Equalizer.m

```
function y = Equalizer( u, coefficients)
%#codegen
% Equalizer using element-by-element multiplication
y = u.*coefficients;
end
```

调用脚本或测试脚本 call _ Equalizer. m，定义了输入变量声明并调用函数得到输出。通过执行这个脚本，我们可以在代码转换前得到预期的值。我们可以用这个值验证代码转换后输出函数是否可以得到正确结果。为了单纯的展示效果，我们用一个任意的函数（余弦函数，取值范围为 1 ~ 100rad）设定系数（coef）与接收数据（u）的采样相乘（即均衡）。

Algorithm

MATLAB testbench: call_Equalizer.m

```
u=1:100;                % First input
coef=cos(u);            % Second input
y=Equalizer(u,coef);    % Function call
```

创建测试脚本的另一个重要原因是：代码转换过程与函数输入定义密切相关。代码转换引擎需要将一个动态类型语言（MATLAB）映射到一个静态类型语言（C/C + +）。这个过程中，数据类型、长度，和每个函数的复变性在生成的 C 代码中明确定义。对用户唯一的要求就是定义函数输入的数据类型、长度，和复变性。代码转换引擎随后将根据输入参数的这些定义确定所有其他内建变量并生成 C 代码。

10. 5. 2　使用 MATLAB 命令

从 MATLAB 函数生成 C 代码最简单的办法是使用 codegen 命令。调用 codegen 时，需要被定义 MATLAB 函数名和命令声明列表。一个重要的命令声明是 args，它定义了所有 MATLAB 工作区内所有函数输入变量实例的长度、类，和复变性。

例如，在函数 Equalizer. m 生成 C 代码时，我们需要首先运行调用函数 call _ Equalizer. m 在 MATLAB 工作区建立函数输入变量 u 和 coef。然后我们在命令行键入 codegen：

Algorithm

```
>> codegen -args {u , coef} Equalizer.m
```

函数 Equalizer. m 是我们生成的 MEX 函数或 C/C + + 可执行代码中的 MATLAB 接口函数。默认情况下，当没有定义其他声明时，codegen 命令生成一个 MEX 函数。生成的 MEX 函数的默认文件名为原函数名附加_ mex 后缀。例如，一个名为 Equalizer _ mex. mex <platform> 的 MEX 函数是相同目录下 MATLAB 接口函数。<platform> 后缀对应操作系统。如，假如 MATLAB 安装在 64 位 Windows 操作系统下，则 MEX 文件全名为 Equalizer _ mex. mexw64。

生成一个 C/C + + 库的命令为在命令后添加 - config 选项：

Algorithm

```
>> codegen -args {u , coef} Equalizer.m -config:lib -report
```

定义代码转换输出类型可以用 - config 选项。用 - config：lib 选项可以生成一个包含 C 源文件和头文件的静态 C/C + + 库。默认情况下，这些文件保存在 MATLAB 接口函数 <fcn _ name> 所在文件目录下。表 10. 1 为 config 选项对应各种不同的代码转换输出类型的参数以及相应的生成文件位置。

表 10. 1　配置选项、输出类型和文件保存地址一览

config 选项	输出类型	生成文件的位置
mex	MEX function	*codegen/mex/ <fcn _ name>*
lib	static C/C + + library	*codegen/lib/ <fcn _ name>*
dll	dynamic C/C + + library	*codegen/dll <fcn _ name>*
exe	static C/C + + executable	*codegen/exe/ <fcn _ name>*

report 选项提供一个到生成文件的超链接。当我们点击这个超链接时，会打开代码转换报告。这个报告包括代码转换的结果。其中有三个标签：MATLAB 代码、调用栈，和 C 代码。在 MATLAB 代码标签栏中，显示 MATLAB 函数。在 C 代码标签栏中，如图 10. 1 所示，为生成的 C 文件。

C 源代码与 MATLAB 接口函数同名（如本例中为 Equalizer. c）。如我们所见，源代码在一个 for 循环内做元素乘积进行均衡。注意在测试脚本中，我们将函数输入数据类型定义为双精度浮点型。因此，生成的 C 代码定义其变量类型为 real _ T，对应了双精度浮点型。

C 代码转换结果中也有两个头文件。一个是 rt nonfinite. h，包含所有 MATLAB 中的类型定义，如 real _ T，和无限 MATLAB 数据如 nan 和 inf。另一个是 Equalizer. h，包括在 C/C + + 调用函数中函数原型需要包含的源文件。在下一个

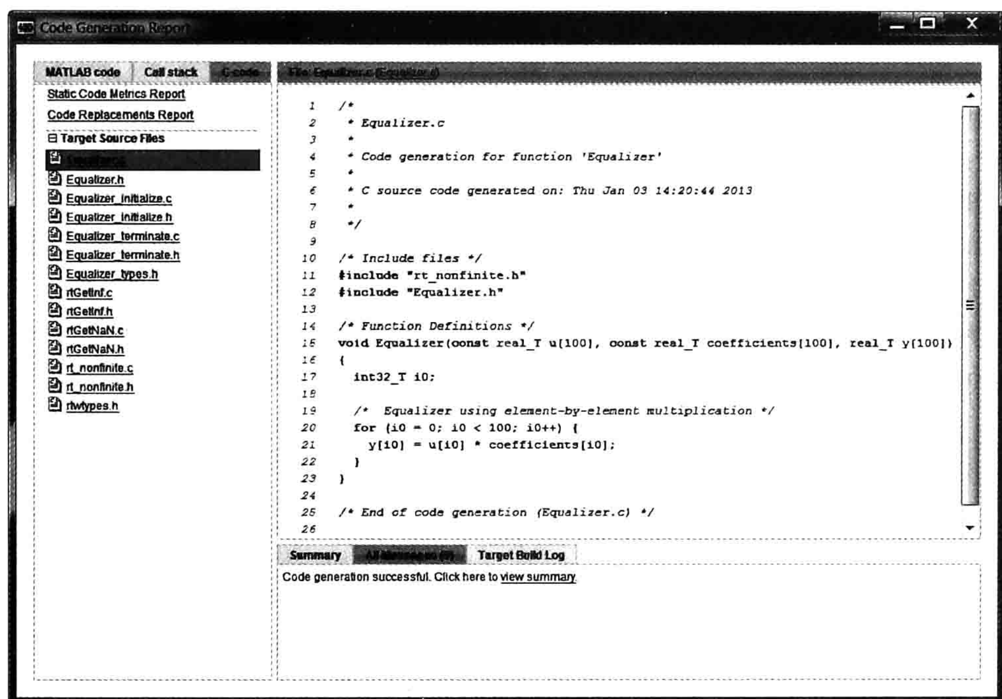


图 10.1 代码转换报告显示生成的 C 代码

子小节中，我们会讨论生成的 C 代码的结构和相应的文件。

10.5.3 使用 MATLAB 代码转换器工程

在本节中，我们将讲解如何使用 MATLAB 代码转换器工程生成 C 代码（见图 10.2）。MATLAB 代码转换器工程是一个 MATLAB 应用程序的实例。它使用图形化用户接口（GUI）和便捷工具辅助代码转换工作。

首先，我们通过键入下面的 MATLAB 命令创建工程：

这个命令创建一个新的 MATLAB 代码转换工程，我们称其为 MyEqualizer。代码转换工程对话框如图 10.3 所示，以目录树的形式表示了工程路径。默认情况下，它设定生成 MEX 函数。下一步是向工程中添加 MATLAB 接口函数。我们既可以将函数拖拽入界面中 Entry - Point Files 区内，也可以通过点击 Add Files 连接进行添加，如图 10.3 所示。

现在，MATLAB 代码转换器已经将文件添加到工程中。这个函数有两个输入参数，u 和 coefficients，显示在文件名之下。注意现在这两个输入变量的数据类型、长度和复变性属性还未定义。为了编译这个函数，我们需要定义脚本以使 MATLAB 代码转换器可以对函数输入变量进行定义。点击 Autodefine Types 连接，

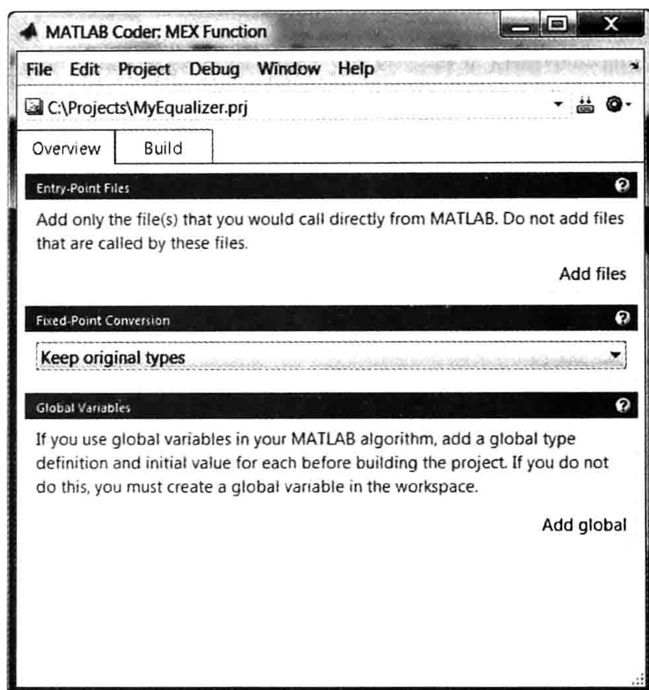


图 10.2　MATLAB 代码转换工程进行代码转换

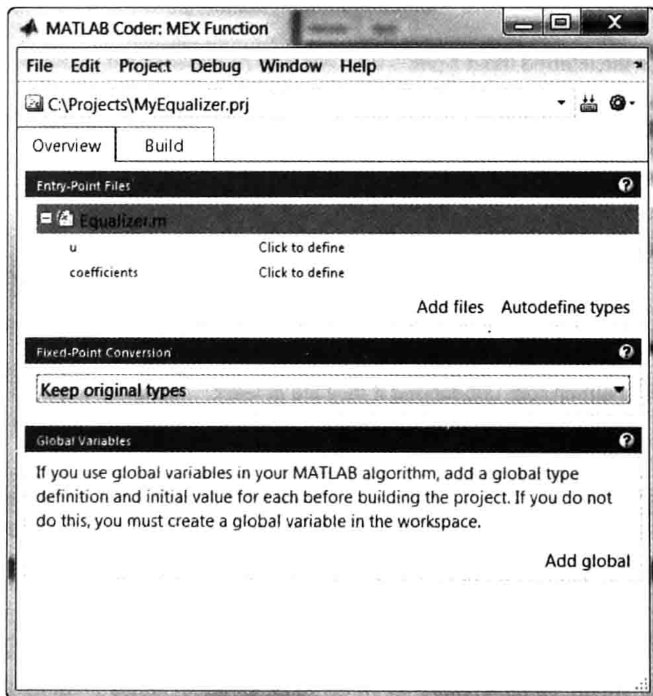


图 10.3　MATLAB 代码转换器中选择 MATLAB 函数

出现自动定义接口输入类型对话框（见图 10.4）。在这个对话框中，我们可以单击“+”按钮添加一个测试文件。这里，我们添加脚本 call _ Equalizer. m。

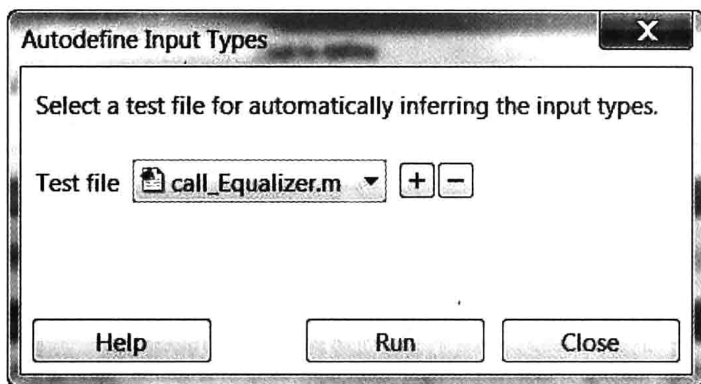


图 10.4　MATLAB 代码转换工程：选择测试脚本的对话框

当我们单击 Run 按钮时，脚本执行。MATLAB 代码转换器对 MATLAB 接口函数的每个输入变量的长度、数据类型，和复变性进行定义。其结果显示在一个叫 Autodefine Input Types 的新的对话框中，如图 10.5 所示。

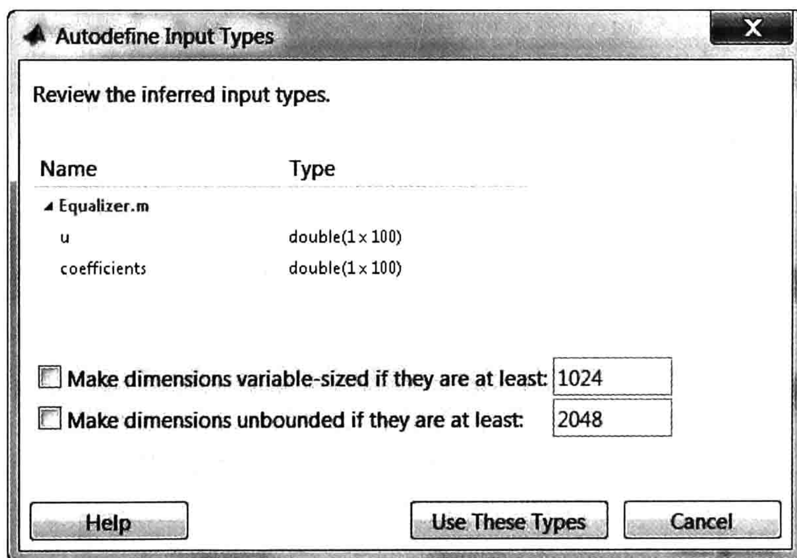


图 10.5　Autodefine Input Types 对话框：从测试脚本中选择数据类型

单击 Use These Types 按钮，我们接收并将其作为输入函数的参数。在最后一步，我们单击 Build 选项卡选择输出函数名和输出类型，并点击 Build 按钮生成代码（见图 10.6）。

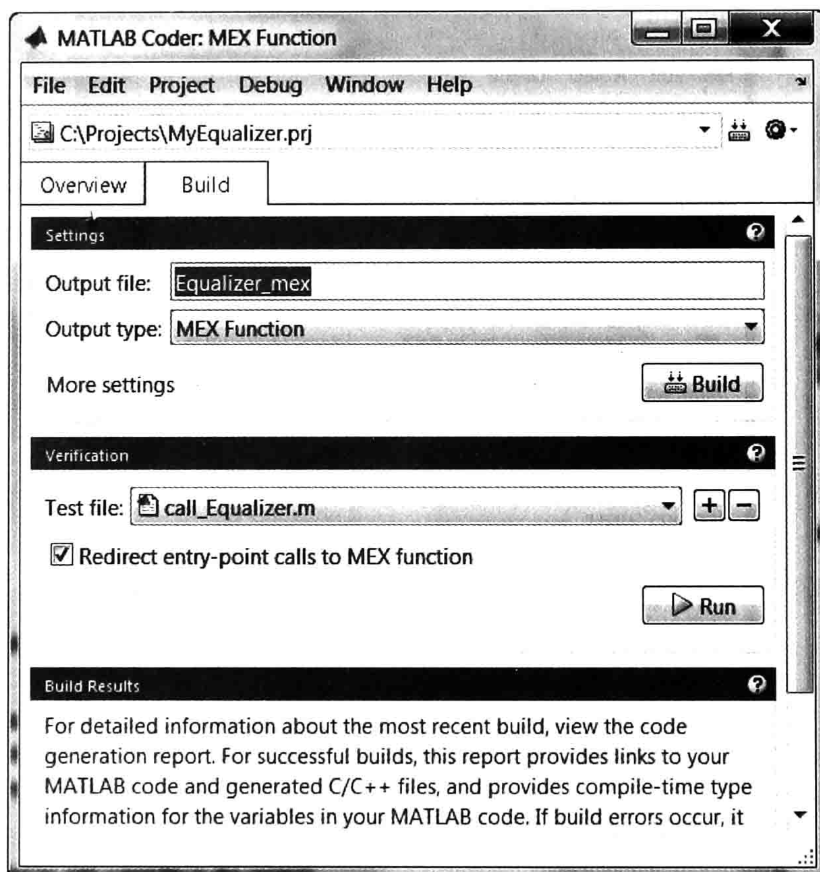


图 10.6 MATLAB 代码转换工程：使用 Build 标签定义输出类型和文件名

默认情况下，输出类型为 MEX 函数。这意味着在代码转换之后，MATLAB 代码转换器将代码编译为只能在 MATLAB 环境调用的 MEX 文件。

MATLAB 代码转换工程的验证功能可以运行 MEX 函数和其定义数据类型的测试脚本（称为脚本）。通过对比 Equalizer. m 函数和 MEX 函数的结果，我们可以验证 MATLAB 函数和生成的 MEX 是否有一样的数值结果。

我们可以通过改变工程的输出类型为动态 C/C + + 库或静态 C/C + + 库得到 C 源代码。在本例中，我们将输出类型改为静态 C/C + + 库并点击 Build 按钮，如图 10.7 所示。在点击 Build 按钮之后，会出现 code – generation Build 对话框（见图 10.8）。如图所示，对话框显示代码转换进程和进程中出现的错误警告信息。

假如代码转换成功，我们可以点击超链接打开代码转换报告，并看到代码转换结果。在本例中，代码转换报告与图 10.1 相同。

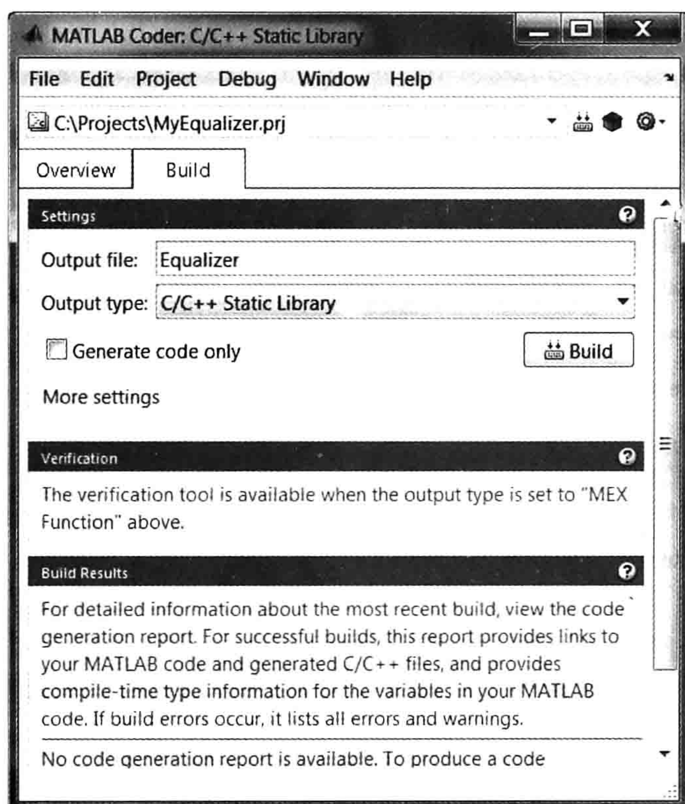


图 10.7　MATLAB 代码转换工程：选择 C/C + +资源代码作为输出类型



图 10.8　MATLAB 代码转换工程：Build 中的代码转换报告

10.6 转换的C代码的结构

转换的C代码有一些预定义的结构。通过代码转换报告中的C代码标签，我们可以看到在C源文件和头文件，它们有和MATLAB接口函数相同的文件名（Equalizer.c和Equalizer.h），还有其他的文件。本例中生成文件的列表如图10.9所示。

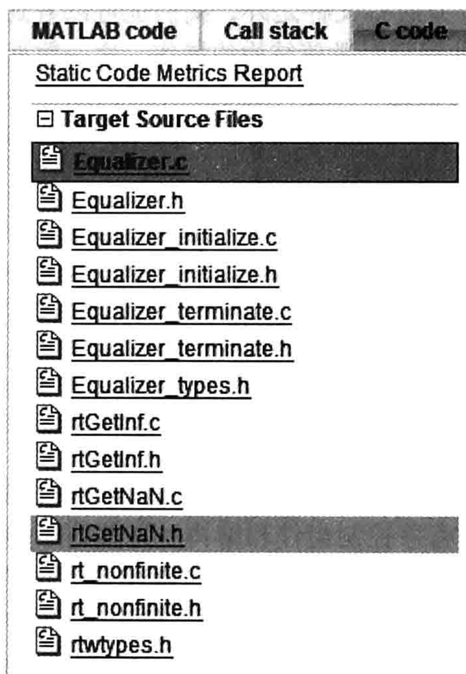


图10.9 代码转换报告：生成文件一览表

仿真中对MATLAB函数的操作可以分为三类：

1）初始化为只执行一次的操作，初始化阶段为只执行一次的操作，初始化在处理循环开始之前。

2）函数调用（或循环内处理）：包括每次函数调用时的处理。

3）终止操作包括仿真终止的操作，包括清理初始化和函数调用时分配的资源。

MATLAB函数的C代码用相同的结构进行不同类型的处理。如均衡器的例子所示：

1）Equalizer_initialize.c和Equalizer_initialize.h对应只在初始化时执行的处理。

2）Equalizer. c 和 Equalize. h 对应循环的主函数调用处理。

3）Equalizer _ terminate. c 和 Equalizer _ terminate. h 对应为终止处理。

对 Equalizer 函数这样按元素处理的简单例子来说，初始化和终止操作的 C 文件（Equalizer _ initialize. c 和 Equalizer _ terminate. c）为空，不包含任何处理。不过对包含常变量或需要初始化的数据这样更复杂的函数来说，初始化函数包含了初始化处理。于此相似，对一些函数，如使用动态内存分配建立变量的，终止函数一般包含 free（）操作将动态分配的内存释放。

除了和各个处理有关的 C 文件之外，我们也可以看到有六个文件进行类型定义和 MATLAB 中“非限定”数值结构的操作。这一数据类型不是 C 默认支持的。它用于算法中有些取值为 nan（非数值）和 inf（无限大）的操作。定义“非限定”的文件包括 rt _ nonfinite. c、rt _ nonfinite. h、rtGetInf. h、rtGetNaN. c，和 rtGetNaN. h。

文件 rtwtypes. h 包括所有所必需的类型信息、宏定义操作，以及 MATLAB 支持的数据类型。根据 MATLAB 函数的不同，也会生成其他不同类型的文件。代码转换和文件划分有关的完整说明，请参考 MATLAB 文档[2]。

10.7 支持的 MATLAB 子集

代码转支持 MATLAB 语言基本子集。支持的特性包括所有标准矩阵操作、多种数据类型，和多种程序控制组件和结构。完整的 MATLAB 语言代码转换支持特性列表请参考 MATLAB 文档[2]，包括：双精度和单精度浮点型、整型，和定点型、复数、字符型、数值类、*N* 阶数组、结构体、矩阵操作、算数运算、关系代数运算和逻辑运算，订阅和函数句柄、永久和全局类型变量、程序控制声明（if、switch、for 和 While 循环）、可变长度数据、可变长度输入和输出声明列表，和 MATLAB 工具箱函数子集，以及 MATLAB 类。

各种工具箱（包括信号处理工具箱）中超过 400 中操作和函数可被支持生成 C 代码。在最新的 MATLAB 版本（R2013a）中，系统工具箱（DSP 系统工具箱、通信系统工具箱和计算机可视化系统工具箱）中超过 300 种系统对象可被支持生成 C 代码。下面的 MATLAB 语言特征不被 C/C + + 代码转换所支持：匿名和嵌套函数、单元数组、Java、递归式、稀疏矩阵，和 try/catch 声明。

10.7.1 代码转换准备

MATLAB 代码转换器支持一些自动工具辅助评估代码转换准备与否。这些工具可以识别哪些 MATLAB 代码不能转换为 C 代码。这些工具可以帮助我们通过一系列详细有针对性的提示消息，更改不支持 C 代码转换的语句，从而成功

完成代码转换。下面我们会讨论如何使用两个辅助工具：MATLAB 代码分析器和代码准备报告。

10.7.2 实例研究：插入导频信号

在本节中我们用一个例子说明辅助工具如何帮助我们识别和更正代码转换问题。这个例子是一个通过“导频”符号的给定集合，找到延所有行和列——即所有子载波和所有子帧正交频分复用（OFDM）符号——插入的均衡器系数的算法。函数（equalizer. m）如小节 5.16 所示。算法的第一个版本 MyInterp0. m 如下：

Algorithm

MATLAB function: MyInterp0.m

```
function out = MyInterp0(y)
%#codegen
UpsampFactor=6;
out=interp(y,UpsampFactor);
```

当我们在 MATLAB 编辑器中编辑函数和注释时，代码分析器会随录入检查代码。它会提示有关代码警告或错误信息并可以更改函数。消息自动更新并提示代码更改可否解决转换问题。

例如，在函数 MyInterp0. m 中，假如我们在一行的结尾调用 interp 函数时用“}”字符而不是“)”时，代码分析器在 MATLAB 编辑器界面提示错误（见图 10. 10)。注意消息栏上端的消息提示是红色的。在第四行，代码分析器显示这条消息对应的单元数组不被代码转换支持。因为单元数组是由{}表示而不是()，故这条消息正确。通过在按右侧语法改正调用函数代码后，这条错误消息将消失，而消息提示会变成绿色。

让我们来看 MATLAB 代码转换器工程如何处理这个函数的代码转换问题。现在输入如下命令：

Algorithm

```
>> coder -new MyInterp
```

当在 MATLAB 接口文件处添加 MyInterp0. m 时，工程对话框会出现如下消息：“View code generation readiness issues（请浏览代码转换准备问题)”。点击链接后，出现如图 10. 11 所示工程代码转换准备报告。这个报告标识了我们算法中不被支持的函数（信号处理工具箱的 interp 函数）。当不被支持的函数由一个可

支持的 MATLAB 函数或特征替换后，准备报告会提示代码转换错误已解决，代码转换可以进行。

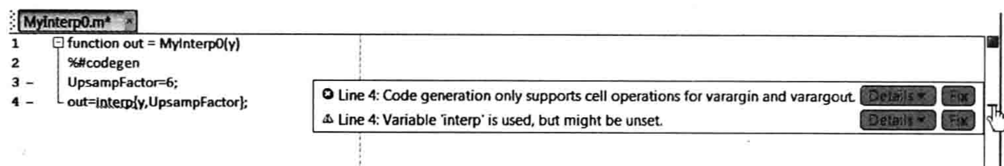


图 10.10　MATLAB 编辑器界面显示代码分析器报告的错误消息

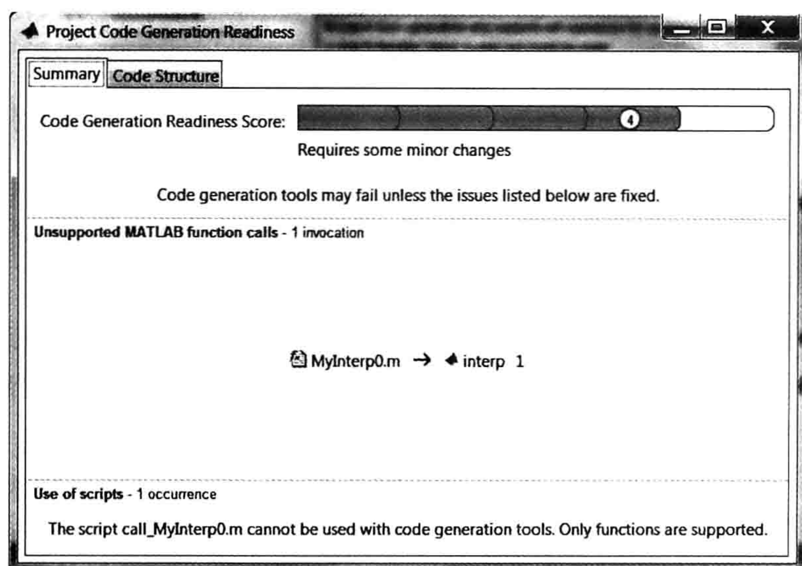


图 10.11　MATLAB 代码转换工程：代码转换准备报告

10.8　复数和本地 C 类型

在本节中，我们用均衡器的例子展示包含复数的算法如何转换成 C 代码。这个均衡器设计为不论输入变量是标量还是向量，或任意大小的矩阵都可执行相同处理的算法。

例如，当输入是一个矩阵时进行代码转换、我们不需要改写算法：我们只需要改写调用函数的测试脚本。在下面的测试脚本 call _ Equalizer2. m 中，新建的输入变量 u 和 coef 都是复数矩阵，它们的阶数为 72 行 14 列，数据类型为单精度浮点型。

Algorithm

MATLAB calling script: call_Equalizer2.m

```
u=complex(single(randn(72,14)));                          % First input
coef= single(randn(72,14)) +1j * single(randn(72,14));    % Second input
y=Equalizer(u,coef);                                      % Function call
```

当我们运行这个测试脚本时，在 MATLAB 工作去创建变量（u，coef 和 y）。键入 MATLAB 命令 whos，我们可以观察到这些变量的大小、类（数据类型）和复变性（见图 10. 12）。

我们可以重复前一节中的步骤将 Equalizer. m 函数转换成 C 代码。首先，单击 MATLAB 代码转换工程中的 Autodefine Tyeps 链接，在一个新的测试脚本中定义输入变量的类型的长度（见图 10. 13）。这时，我们可以选择数据类型为复单精度浮点型矩阵，矩阵大小为 72 × 14，如图 10. 14 所示。最后，在 Build 标签输出类型处选择 C/C + + 静态库，我们就可以生成 Equalizer 函数的 C 代码。

图 10. 15 所示为代码转换的输出。注意输入变量在 C 代码中定义为一种新类型 creal32 _ T，对应单精度浮点型的复变量。所有这一类型的定义，都由 MATLAB 代码转换器自动生成，并可以在 rtwtypes. h 文件中找到。这个文件包含了所必须的类型信息和定义复变量操作的宏。实际上，MATLAB 支持的所有操作和所有数据类型都可一定程度上转换为 C 代码。头文件和生成的 C 文件体现了这种一对一的关系。

```
>> whos
  Name        Size            Bytes  Class     Attributes

  coef        72x14            8064  single    complex
  u           72x14            8064  single    complex
  y           72x14            8064  single    complex
```

图 10. 12　观察函数输入变量的数据类型、长度和复变性

注意 MATLAB 的按元素矩阵乘法如何用 for 循环的元素数组乘法表示。因为两个输入矩阵为复数，在生成的 C 源代码中，按元素运算分为实部（. re）和虚部（. im）分别进行。

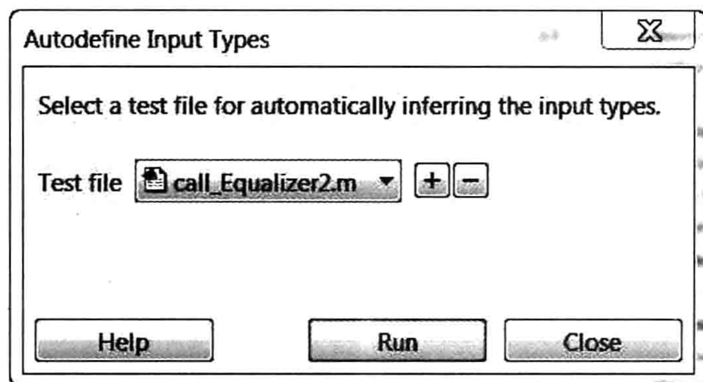


图 10.13　MATLAB 代码转换工程：选择测试脚本测试生成的 MEX 函数

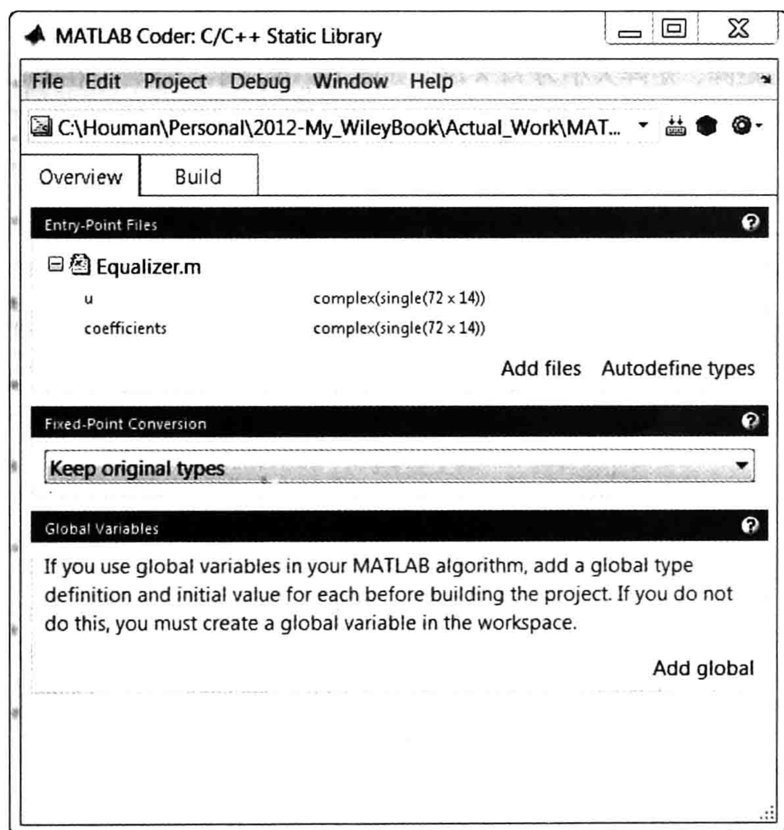


图 10.14　MATLAB 代码转换工程：用测试文件改变输入变量的数据类型

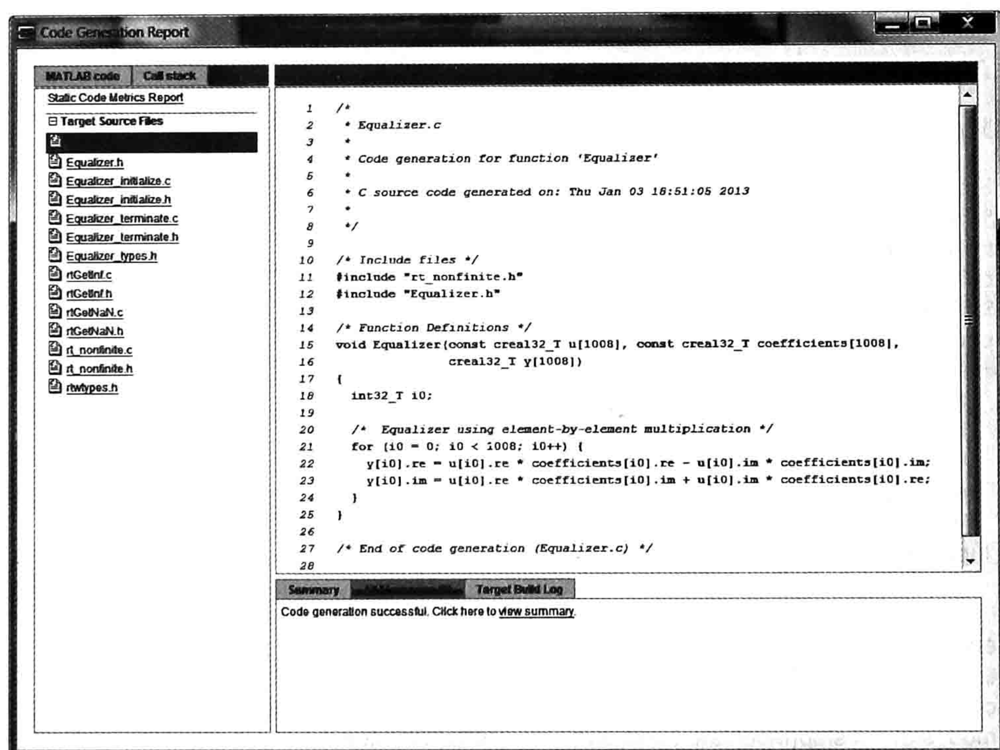


图 10.15　代码转换报告：生成包含复数数据类型输入的源代码

10.9　系统工具箱支持

MATLAB 操作和函数的主要子集都支持代码转换。另外，DSP 和通信系统工具箱的系统对象和信号处理工具箱也支持代码转换。

在本节中我们会使用系统工具箱中的系统对象作为代码转换的例子。用系统工具箱进行代码转换有两点好处：首先，系统工具箱对 C 代码进行了各种优化；其次，通过借助系统工具箱实现算法，我们可将更多时间用于架构系统组件而不是重写和优化算法组件。

10.9.1　实例研究：FFT 和反 FFT

下面的函数是一个简单收发端例子。在发射端，输入比特经过调制和反快速傅里叶变换（IFFT）生成调制符号，随后通过信道模型和加性高斯噪声（AWGN）信道。在接收端，首先进行傅里叶变换（FFT），随后进行解调生成输出比特。通过对比输入和输出比特，我们可以计算比特误码率（BER）。本例中使用通信系统工具箱中如下系统对象：调制器、解调器、卷积编码器、Viterbi 译

码器、循环冗余校验（CRC）生成器、CRC 校验器，和 AWGN 信道。

Algorithm

MATLAB function

```
function y=Transceiver0(u)
%% Constants
trellis=poly2trellis(7, [133 171]);
polynomial=[1 1 zeros(1, 16) 1 1 0 0 0 1 1];
%% Initializations
persistent Modulator DeModulator ConvEncoder Viterbi CRCGen CRCDet
if isempty(Modulator)
   Modulator      = comm.QPSKModulator('BitInput',true);
   DeModulator  =  comm.QPSKDemodulator('BitOutput',true);
   ConvEncoder  =  comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'TrellisStructure', trellis);
   Viterbi            = comm.ViterbiDecoder('TrellisStructure', trellis,
'InputFormat','Hard','TerminationMethod','Truncated');
   CRCGen          = comm.CRCGenerator('Polynomial', polynomial);
   CRCDet           = comm.CRCDetector   ('Polynomial', polynomial);
end
tb             = step(CRCGen , u);                   % CRC generator
cod_sig      = step(ConvEncoder , tb);            % Convolutional encoder
mod_sig     = step(Modulator, cod_sig);         % QPSK Modulator
sig            = ifft(mod_sig);                         % Perform IFFT
rec            = fft(sig);                                  % Perform FFT
demod       = step(DeModulator, rec);             % QPSK Demodulator
dec           = step(Viterbi , demod);               % Viterbi decoder
y               = step(CRCDet , dec);                   % CRC detector
```

代码转换过程和上一例相同，结果显示在代码转换报告中。更复杂的算法可以使用工具箱组件构建。这意味着生成的 C 文件的大小更大，全部的 C 代码无法在这里显示；图 10.16 只显示了前面的几行 C 代码。

Algorithm

MATLAB function

```
function [u, y]=Transceiver1
%% Constants
trellis=poly2trellis(7, [133 171]);
polynomial=[1 1 zeros(1, 16) 1 1 0 0 0 1 1];
%% Initializations
persistent Modulator DeModulator ConvEncoder Viterbi CRCGen CRCDet
if isempty(Modulator)
   Modulator      = comm.QPSKModulator('BitInput',true);
   DeModulator  =  comm.QPSKDemodulator('BitOutput',true);
```

```
   ConvEncoder  = comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'TrellisStructure', trellis);
   Viterbi        = comm.ViterbiDecoder('TrellisStructure', trellis,
'InputFormat','Hard','TerminationMethod','Truncated');
   CRCGen         = comm.CRCGenerator('Polynomial', polynomial);
   CRCDet         = comm.CRCDetector   ('Polynomial', polynomial);
end
u              = randi([0 1], 2024,1);             % Random bits generation
tb             = step(CRCGen , u);                  % CRC generator
cod_sig        = step(ConvEncoder , tb);            % Convolutional encoder
```

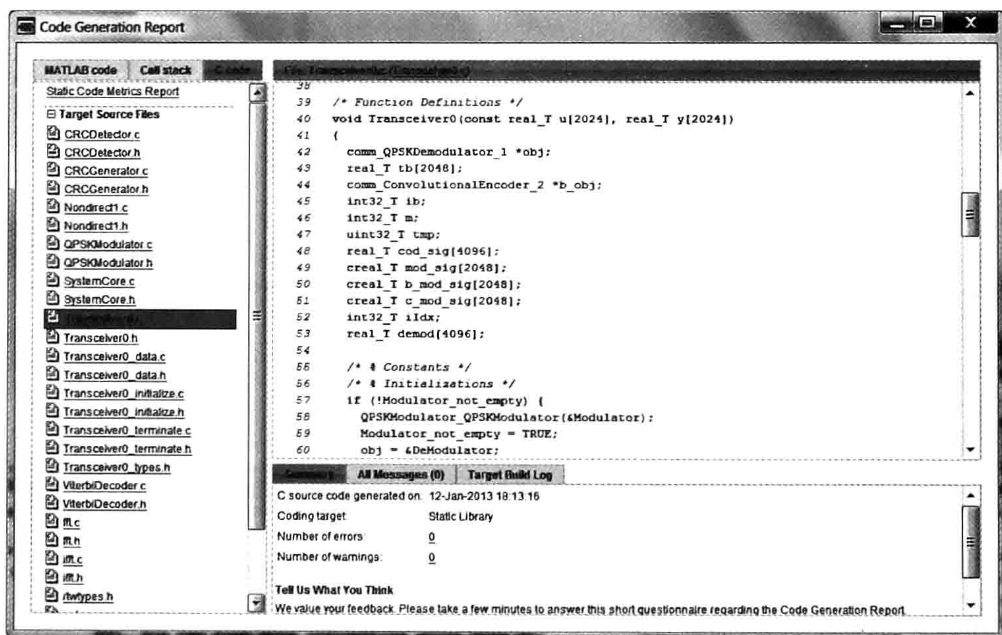


图 10.16　代码转换报告：生成代码的其中几行

我们首先关闭“非限定”数据类型进行优化，以减小生成 C 代码的大小。MATLAB 代码转换工程的用户自定义页有一个 Speed 标签。我们可以通过勾销相应的选项关闭“非限定”数据类型支持，如图 10.17 所示。我们也可以观察到，算法并不包含表示分支和存储的变量。所以生成的 C 代码中初始化函数（transceiver _ initialize. c）如图 10.18 所示。

我们观察算法包含状态变量对转换后 C 代码的影响。在 C 代码结尾处，我们添加一个随机比特生成器生成输入比特。因为随机数生成器与核或状态保持有关，所以生成的 C 代码中会有相当数量的初始化代码。

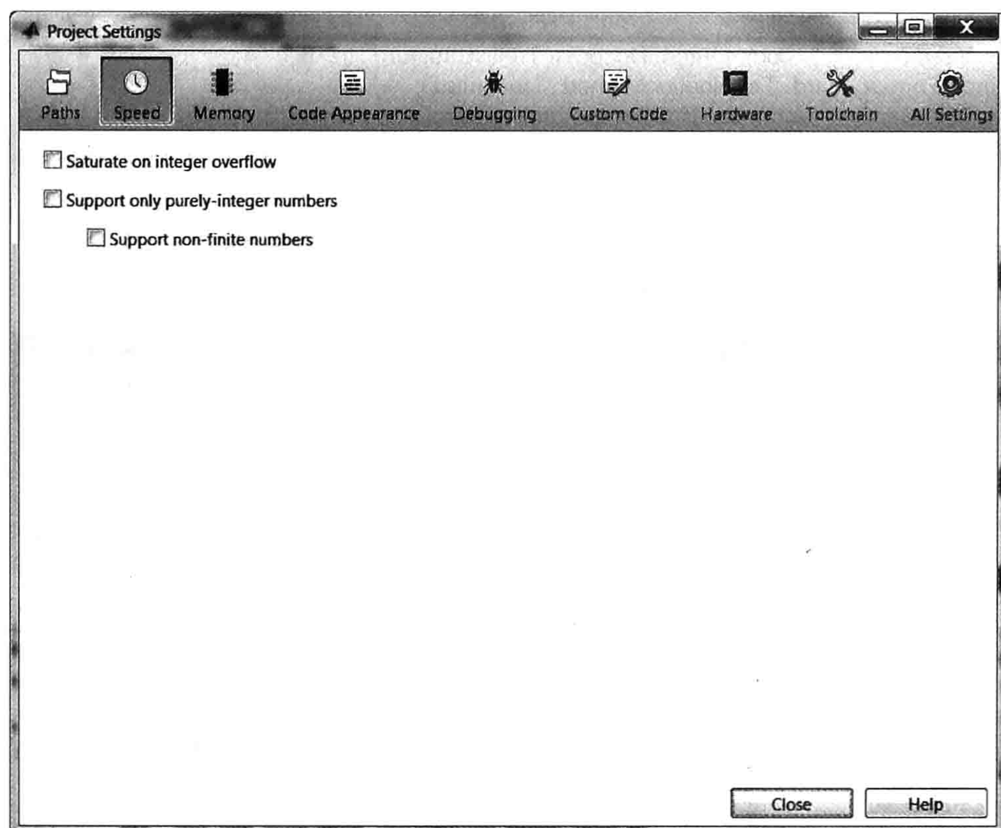


图 10.17 MATLAB 代码转换工程：与仿真速度相关的选项

```
mod_sig   = step(Modulator, cod_sig);      % QPSK Modulator
sig       = ifft(mod_sig);                 % Perform IFFT
rec       = fft(sig);                      % Perform FFT
demod     = step(DeModulator, rec);        % QPSK Demodulator
dec       = step(Viterbi , demod);         % Viterbi decoder
y         = step(CRCDet , dec);            % CRC detector
```

在更新的函数（Transceiver1. m）中，我们用 MATLAB 的 randi 生成输入比特。因此，函数没有输入而有两个输出。生成的初始化代码文件 transceiver _ initialize. c 展示了包含状态的算法如何初始化。注意 C 代码的初始化算法更新了变量 b _ state，其定义为一个静态变量。在 C 代码中使用静态变量是表示一个变量在多个调用中维持某值并实现某个状态的办法（见图 10.19 和图 10.20）。

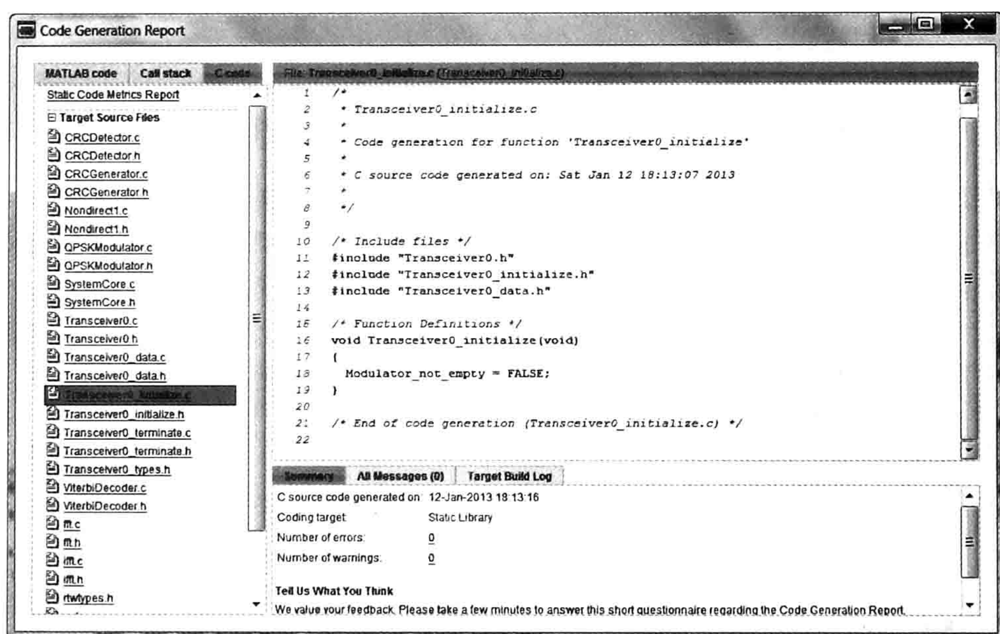


图 10.18 代码转换报告：初始化函数

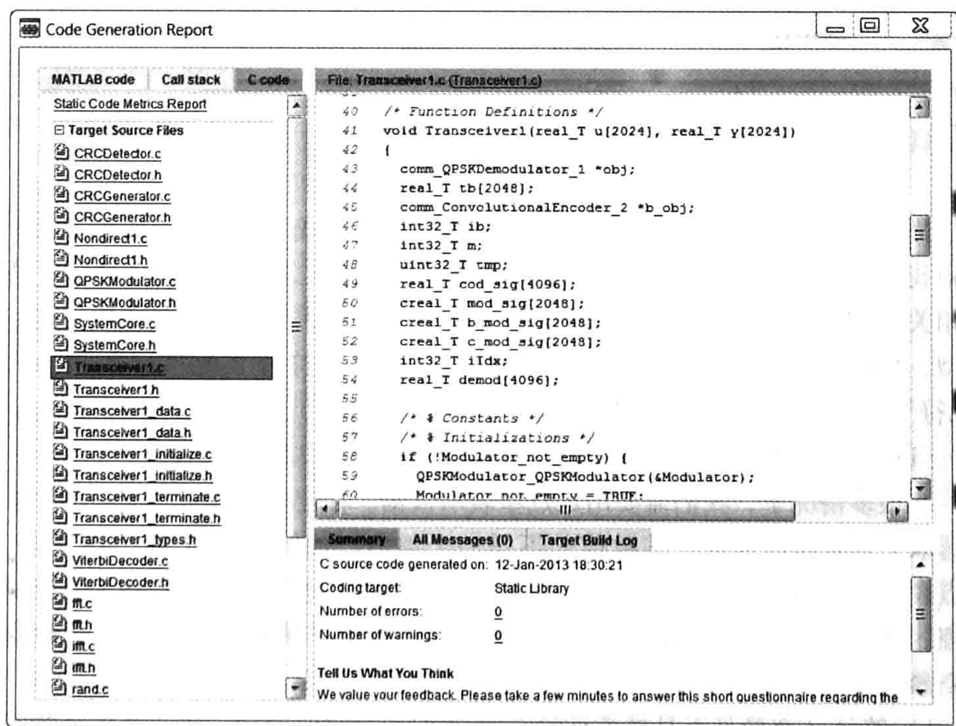


图 10.19 Transceiver1. m 转换后的代码，显示了生成 C 代码的前几行

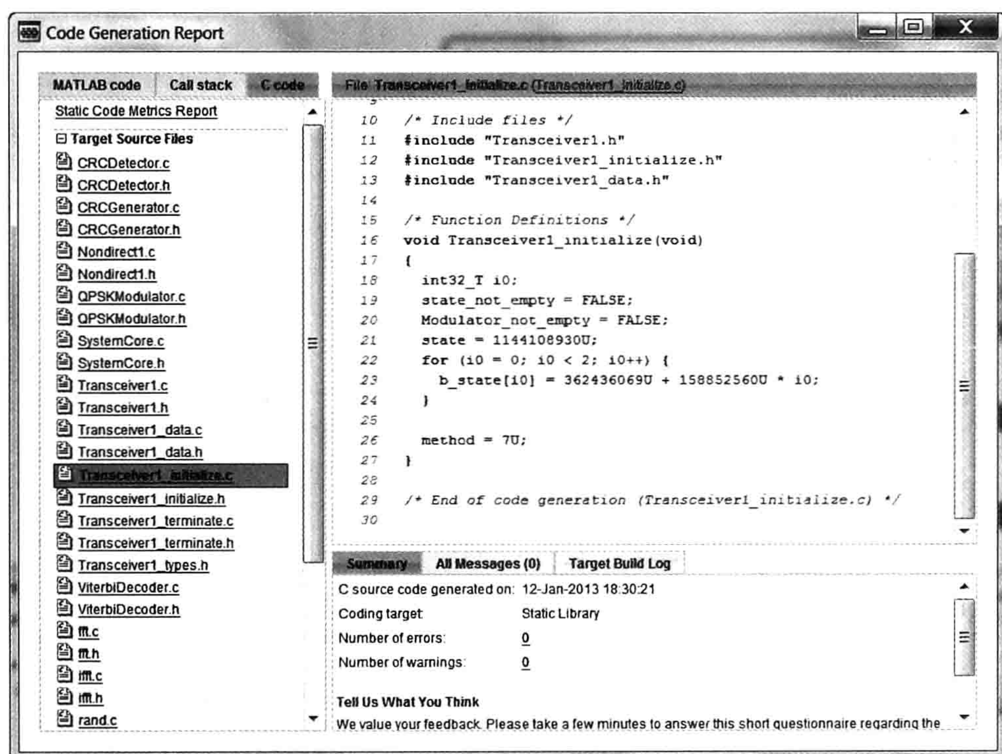


图 10.20　新的初始化函数——设置了随机数生成器的核

10.10　定点型数据支持

目前，本章中涉及的函数处理的数据都是单精度或双精度浮点类型。在上一节中我们也引入了几个 MATLAB 函数和工具箱生成二进制数据表示发射比特。其相关变量用布尔型表示。在很多情况下，函数中的表示序数和量化值一般使用整型。MATLAB 支持六种不同的整型数：uint8（无符号 8 位整型数）、uint16（无符号 16 位整型数）、uint32（无符号 32 位整型数）、int8（带符号 8 位整型数）、int16（带符号 16 位整型数），和 int32（带符号 32 位整型数）。

在很多情况下，我们需要用定点型表示数据。在定点算数中，从一个有限集中选取一定范围的取值。实数值并不是一个整数。如第二章所述，MATLAB 用定点表示和定点算数声明的变量可以用定点型设计器（也叫做定点型工具箱）处理。MATLAB 中的任何定点型可以表示为 fi 对象。我们需要定义这个对象的三个参数：

1）符号（变量是否是带符号的）；

2）字长（由多少比特表示一个数）；

3）小数部分的长度（用多少比特表示一个数的小数部分）。显然，整数（一个数的整数部分）等于字长减去表示小数部分和符号部分的比特。

10.10.1 实例研究：FFT 函数

在本节中，我们对上一个例子（函数 Transceiver0. m）更新用定点型类型表示正交相移监控（QPSK）调制的结果。因为调试前所有变量的数据类型为布尔型，假如调制器变为定点型并传递和进行反 FFT 运算，则全部函数都将使用定点型和整型数据类型。算法的第一个版本如下所示。

Algorithm

MATLAB function

```
function y=Transceiver0_fixed(u)
%% Constants
trellis=poly2trellis(7, [133 171]);
polynomial=[1 1 zeros(1, 16) 1 1 0 0 0 1 1];
%% Initializations
persistent Modulator DeModulator ConvEncoder Viterbi CRCGen CRCDet
if isempty(Modulator)
   Modulator      = comm.QPSKModulator('BitInput',true,'OutputDataType','Custom');
   DeModulator  = comm.QPSKDemodulator('BitOutput',true);
   ConvEncoder  = comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'TrellisStructure', trellis);
   Viterbi           = comm.ViterbiDecoder('TrellisStructure', trellis,
'InputFormat','Hard','TerminationMethod','Truncated');
   CRCGen         = comm.CRCGenerator('Polynomial', polynomial);
   CRCDet          = comm.CRCDetector   ('Polynomial', polynomial);
end
tb            = step(CRCGen , u);                   % CRC generator
cod_sig     = step(ConvEncoder , tb);          % Convolutional encoder
mod_sig    = step(Modulator, cod_sig);        % QPSK Modulator
sig           = ifft(mod_sig);                      % Perform IFFT
rec          = fft(sig);                               % Perform FFT
demod     = step(DeModulator, rec);           % QPSK Demodulator
dec          = step(Viterbi , demod);            % Viterbi decoder
y              = step(CRCDet , dec);               % CRC detector
```

可以通过两个途径将函数转为定点型：我们可以定义调制器系统对象的输出数据类型为定点型并定义其内容；或者我们可以用 fi 对象在双精度浮点型数据生成之后，建立定点型调制器输出。第二种方法的向量在 MATLAB 工作区中同时有浮点型和定点型两个版本，但消耗内存。在解调之后，我们使用的硬判决译码的输出为布尔型，也因为硬判决译码返回比特序列。

我们可以运行测试脚本并转换代码。当我们考察 MATLAB 代码转换工程时，

准备报告提示 ifft 和 fft 函数不支持定点型作为函数输入。为了能支持定点型类型算法，我们现在必须找到一个替代算法支持定点型传递和进行反 FFT 运算。DSP 系统工具箱的系统对象 dsp. FFT 和 dsp. IFFT 可以满足我们的要求。这个例子也明确了为什么信号处理工具箱和 DSP 系统工具箱有这两个看似多余的组件。DSP 系统工具箱有更多面向实现的组件和更多对定点型类型算法的支持，它们面向用户关心的硬件实现。DSP 系统工具箱的 dsp. FFT 和 dsp. IFFT 支持定点型类型鲜明地体现了工具箱支持算法的托管实例，以及最终硬件实现的需要。通过用 DSP 系统工具箱的 dsp. FFT 和 dsp. IFFT 系统对象分别替换 fft 和 ifft，函数可以处理定点型数据。

Algorithm

MATLAB function

```
function y=Transceiver0_fixed2(u)
%% Constants
trellis=poly2trellis(7, [133 171]);
polynomial=[1 1 zeros(1, 16) 1 1 0 0 0 1 1];
%% Initializations
persistent Modulator DeModulator ConvEncoder Viterbi CRCGen CRCDet FFT IFFT
if isempty(Modulator)
   Modulator     = comm.QPSKModulator('BitInput',true,'OutputDataType','Custom');
   DeModulator  =  comm.QPSKDemodulator('BitOutput',true, 'OutputDataType',
'Smallest unsigned integer');
   ConvEncoder  =  comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'TrellisStructure', trellis);
   Viterbi          = comm.ViterbiDecoder('TrellisStructure', trellis, 'InputFormat','Hard',...
      'TerminationMethod','Truncated', 'OutputDataType','logical');
   CRCGen         = comm.CRCGenerator('Polynomial', polynomial);
   CRCDet          = comm.CRCDetector   ('Polynomial', polynomial);
   FFT  = dsp.FFT;
   IFFT = dsp.IFFT;
end
tb             = step(CRCGen , u);                  % CRC generator
cod_sig      = step(ConvEncoder , tb);             % Convolutional encoder
mod_sig     = step(Modulator, cod_sig);          % QPSK Modulator
sig             = step(IFFT, mod_sig);                 % Perform IFFT
rec            = step(FFT, sig);                          % Perform FFT
demod        = step(DeModulator, rec);            % QPSK Demodulator
dec            = step(Viterbi , demod);               % Viterbi decoder
y               = step(CRCDet , dec);                   % CRC detector
```

MATLAB 代码转换器随后可以成功的转换函数代码。

图 10. 21 所示为函数的代码转换报告。生成的 C 代码只有整型数据类型。我们注意到其中并没有浮点型变量。也可注意到对每个算数运算，包括量化、饱和，和打包，使用了内联函数。

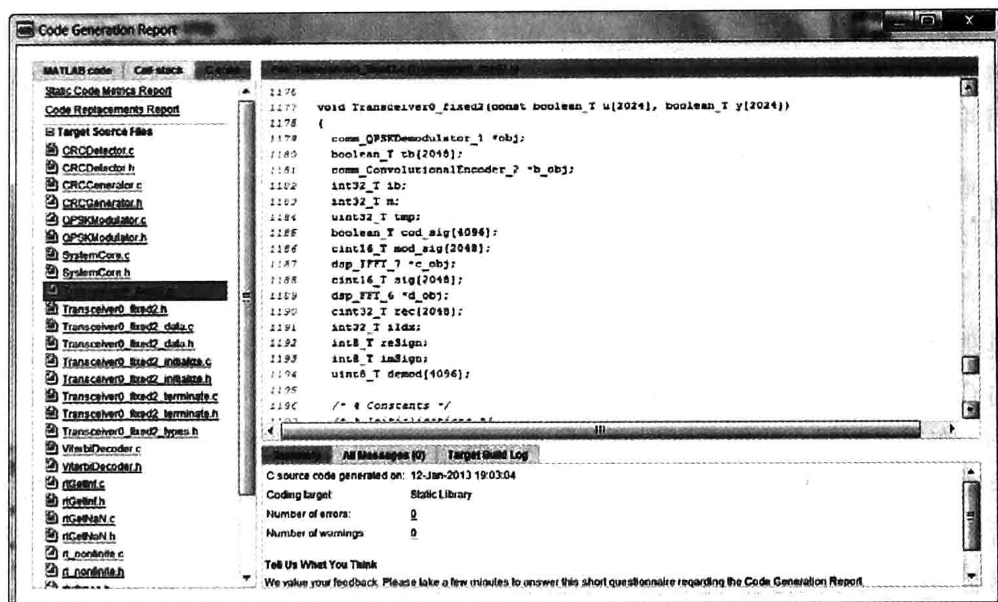


图 10.21　代码转换报告：定点型版本收发端的整型数 C 代码

一个内敛函数的一段代码如图 10.22 所示。这个函数对整型数据进行算数和

```
static void MultiWordSub(const uint32_T u1[], const uint32_T u2[], uint32_T y[],
  int32_T n)
{
  int32_T i;
  uint32_T u1i;
  uint32_T borrow = 0U;
  uint32_T yi;
  int32_T borrow1;
  int32_T borrow2;
  for (i = 0; i < n; i++) {
    u1i = u1[i];
    yi = (u1i - u2[i]) - borrow;
    y[i] = yi;
    if (yi >= u1i) {
      borrow1 = 1;
    } else {
      borrow1 = 0;
    }

    if (yi > u1i) {
      borrow2 = 1;
    } else {
      borrow2 = 0;
    }

    if (borrow != 0U) {
      borrow = (uint32_T)borrow1;
    } else {
      borrow = (uint32_T)borrow2;
    }
  }
}

static void c_MWDSPCG_R2BRScramble_OutPlace(cint16_T y[], const cint16_T x[],
```

C source code generated on: 12-Jan-2013 18:03:04
Coding target: Static Library
Number of errors: 0
Number of warnings: 0

图 10.22　定点型代码转换中多字、基于整型的自动开发过程实例

逻辑运算实现定点型算数运算。这些底层定点型运算假如手动编写，将会花费非常大的设计时间。通过定点型工具箱和 MATLAB 代码转换器，我们可以节约时间并避免很多关键但困难的工作，如将浮点型转换成定点型表示。

10.11 可变长度数据支持

目前，MATLAB 转换的代码中输入数据的长度是固定的。固定长度代码转换非常直接；所有工作只需要定义每个函数的输入长度，即可得到同样长度数据的 C 代码。

在很多时候，我们需要生成在仿真中输入长度可变的代码。例如，在自适应性编码中，随着编码率的变化，信道编码器的输出长度发生着改变，这意味着随后的绕码器和调制器输入长度也需要改变。与此相似，在自适应性调制中，虽然调制器的输入比特数一定，但根据采用 QPSK、16QAM 或 64QAM 调制方案的不同，输出长度也会变化。

在本节中，我们表现 C 代码转换如何适应可变的变量长度。我们一般针对数据长度分三种代码转换模式：

1）定长数据；

2）可变长度数据并限定长度上限；

3）不限定边界的可变长度数据。每种模式都反映了计算复杂度、内存占有率，和灵活性的折中。

10.11.1 实例研究：自适应性调制

作为例子，我们将通过自适应性调制器，展示三种数据长度模式以及它们对代码转换的影响。

Algorithm

MATLAB function

```
function y=Modulator(u)
persistent QPSK
if isempty(QPSK)

   QPSK        = comm.PSKModulator(4, 'BitInput', true, 'PhaseOffset', pi/4, ...
      'SymbolMapping', 'Custom', 'CustomSymbolMapping', [0 2 3 1]);
end
y=step(QPSK, u);
```

让我们考察一个简单的 LTE QPSK 调制器函数（Modulator. m）。这个 MATLAB 函数对输入长度的限定很宽泛。例如，QPSK 调制器的输入长度为偶数，输出长度则为输入的一半。

图 10.23 所示为我们如何执行调制器方程，首先设定输入比特向量长度为 4200×1，随后设定为 256×1。我们用 MATLAB 函数 whos 观察输入和输出长度。不进行代码转换时，函数 Modulator. m 如预期对长度变化的输入进行处理。当输入长度改变时，函数生成的输出长度也相应变化。但我们将在下面看到，当进行代码转换后，生成的 MEX 函数会在输入长度改变时有不一样的行为。

```
>> u=randi([0 1], 4200, 1);
>> y=Modulator(u);
>> whos u y
  Name          Size                Bytes  Class     Attribute

  u          4200x1                 33600  double
  y          2100x1                 33600  double    complex

>> u2=randi([0 1], 256, 1);
>> y=Modulator(u2);
>> whos u2 y
  Name          Size                Bytes  Class     Attributes

  u2         256x1                  2048   double
  y          128x1                  2048   double    complex
```

图 10.23　在不同输入长度设定下调用调制器函数

10.11.2　定长代码转换

下一步，我们输入下面的命令建立一个新的代码转换工程：

Algorithm

```
>> coder -new Modulator
```

在添加 Modulator 函数到工程之后，我们可以在 Autodefine Type 标签中调用下面的 MATLAB 脚本定义长度和数据类型：

Algorithm

MATLAB script

```
u=randi([0 1], 4200,1);
y=Modulator(u);
```

运行脚本后，Autodefine Type 工具正确生成 4200 × 1 长度的函数输入，如图 10.24 所示。我们可以看到窗口中有两个复选框。它们代表当输入函数长度改变时决定生成 MEX 函数的各个操作。假如我们不勾选这两个复选框，我们就以定长代码转换模式工作。我们下面将看到，通过勾选第一个复选框，我们工作在限定上限的可变长代码转换模式。当勾选第二个复选框时，则为无边界限定可变长度代码转换模式。下面我们将讨论细节。

为了充分理解什么是“定长代码转换”，我们不勾选两个复选框，并跳到 Build 标签卡（见图 10.25），生成 MEX 函数。我们以默认名 Modulator _ mex 命名输出的 MEX 函数。

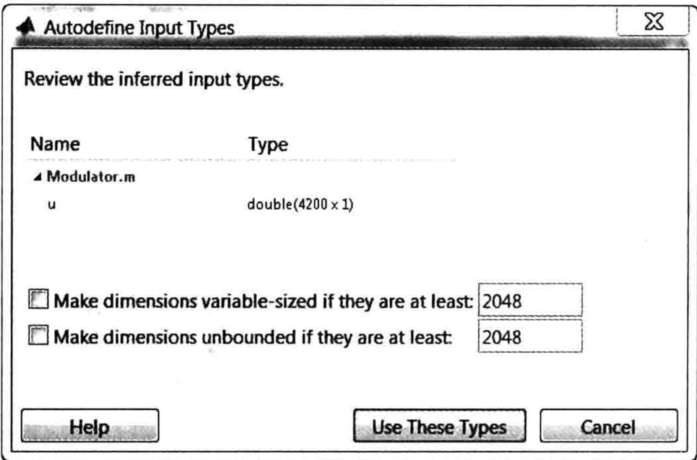


图 10.24　定长代码转换的选项

假如我们调用这个生成的 MEX 函数处理不为 4200 × 1 长度的输入时，会弹出错误信息。在下面的例子中，我们首先调用函数处理正确的输入长度和数据类型（4200 × 1 长度的双精度向量），然后处理 256 × 1 长度的双精度向量。弹出的错误信息如图 10.26 所示。

如我们所见，定长代码转换只能处理特定长度的输入。我们也可以用 codegen 命令行脚本进行定长代码转换，生成调制器的 MEX 函数：

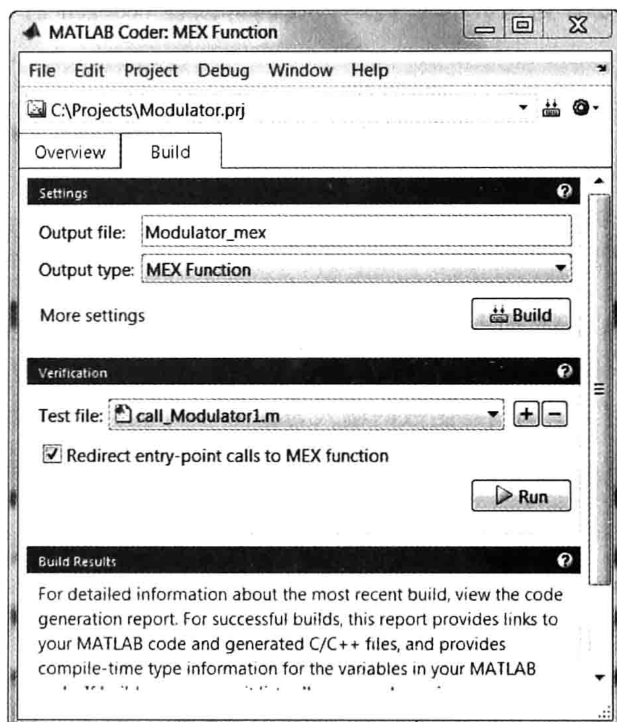


图10.25 MATLAB代码转换工程：编译定长MEX函数

```
>> u=randi([0 1], 4200, 1);
>> y=Modulator_mex(u);
>> whos u y
  Name          Size            Bytes  Class     Attributes

  u          4200x1             33600  double
  y          2100x1             33600  double    complex

>> u2=randi([0 1], 256, 1);
>> y=Modulator_mex(u2);
MATLAB expression 'u' is not of the correct size: expected [4200x1] found
[256x1].

Error in Modulator_mex
```

图10.26 调用调制器的定长MEX函数

Algorithm

MATLAB script

```
u=randi([0 1], 4200,1 );
codegen Modulator -args {u}
```

假如我们想要看到函数的 C 代码，我们可以在 codegen 命令后添加几个参数，如下所示：

Algorithm

MATLAB script

```
u=randi([0 1], 4200,1 );
codegen Modulator -args {u} –config:lib -report
```

通过点击 MATLAB 命令行中的 View Report 链接我们可以看到生成的 C 源代码。当我们打开代码转换报告时，如图 10.27 所示，我们可以看到 C 文件（Modulator. c）定义了调制器函数的输入为有 4200 个元素的常量实数数组。

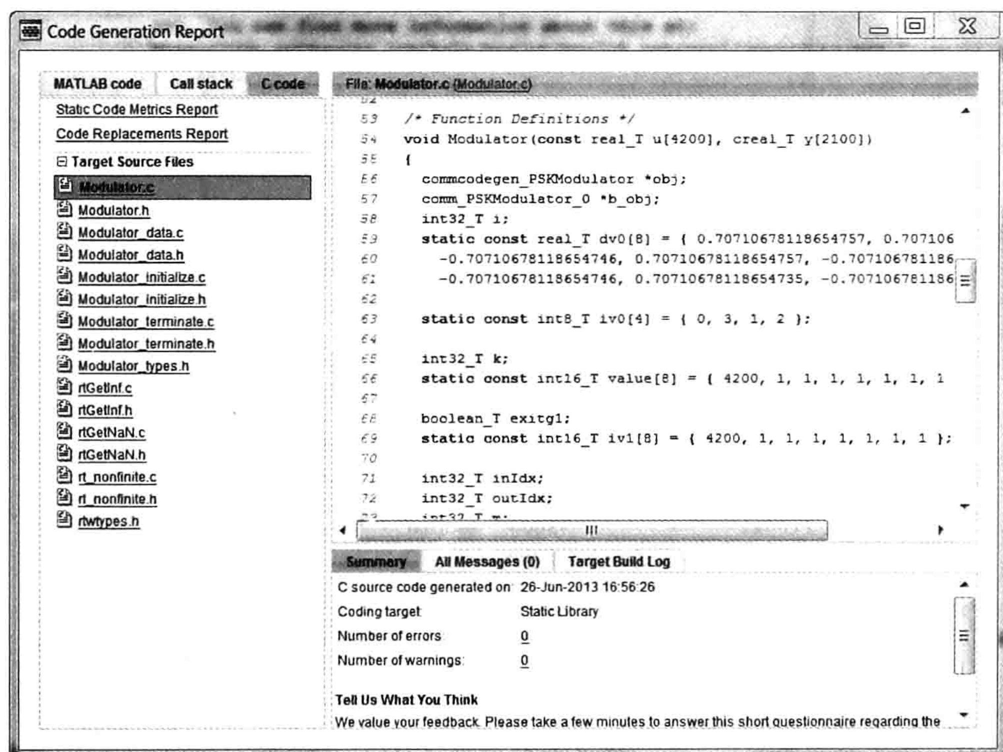


图 10.27　代码转换报告：调制器模型的定点型代码

另外一种方法也可确认定长代码转换是根据其他值定义一个变量的长度的。如果一个值是常数，则代码转换引擎可以根据这个值规定其他变量的长度。例如，在 Modulator _ fixedsize. m 函数中，变量 N 的值决定了对应调制器输出比特的变量 u 的长度。Modulator _ fixedsize. m 函数没有输入，其所有变量都为本地变量。因此，对 Modulator _ fixedsize 进行代码转换的 MATLAB 命令简单的为：

Algorithm

>> codegen Modulator_fixedsize

Algorithm

MATLAB function

```
function y=Modulator_fixedsize
N=4200;
persistent QPSK
if isempty(QPSK)
  QPSK      = comm.PSKModulator(4, 'BitInput', true, 'PhaseOffset', pi/4, ...
    'SymbolMapping', 'Custom', 'CustomSymbolMapping', [0 2 3 1]);
end
u=randi([0 1], N,1);
y=step(QPSK, u);
```

生成的函数 C 源代码和前面图 10.27 中所示相同。

10.11.3 有界变长数据

我们可以通过改变一个代码转换选项对调制器函数进行有界可变长度代码转换。在 MATLAB 代码转换工程中，我们只需要勾选“Make dimensions variable - sized if they are at least（规定可变长度至多为）”，如图 10.28 所示。我们需要对输入长度设定上边界。通过设定最大长度 4200，我们可以以此上边界进行可变长度代码转换。

为了用命令行函数 codegen 得到相同的结果，我们可以使用 coder. typeof 参数，定义函数输入的第一阶为如 4200。

Algorithm

MATLAB script

```
MaxSize = 4200;
u=randi([0 1], MaxSize,1);
codegen Modulator -args {coder.typeof(0,[MaxSize 1],1)}
```

生成的 MEX 函数 Modulator _ mex 可以处理最大长度 4200 以内的输入，如图 10.29 所示。注意当输入长度大于 4200 时将弹出错误。

另一种实现有界可变长度 C 代码的方法为使用 assert 函数。我们现在在调制器函数内假如比特生成函数 randi 实验这种方法。我们调用更新的函数 Modulator _ varsize _ bounded。输入变量 N 决定了调制器输入和输出的长度。为了生成有界可变长度 C 代码，我们用 assert 函数设定变量 N 的取值上限。

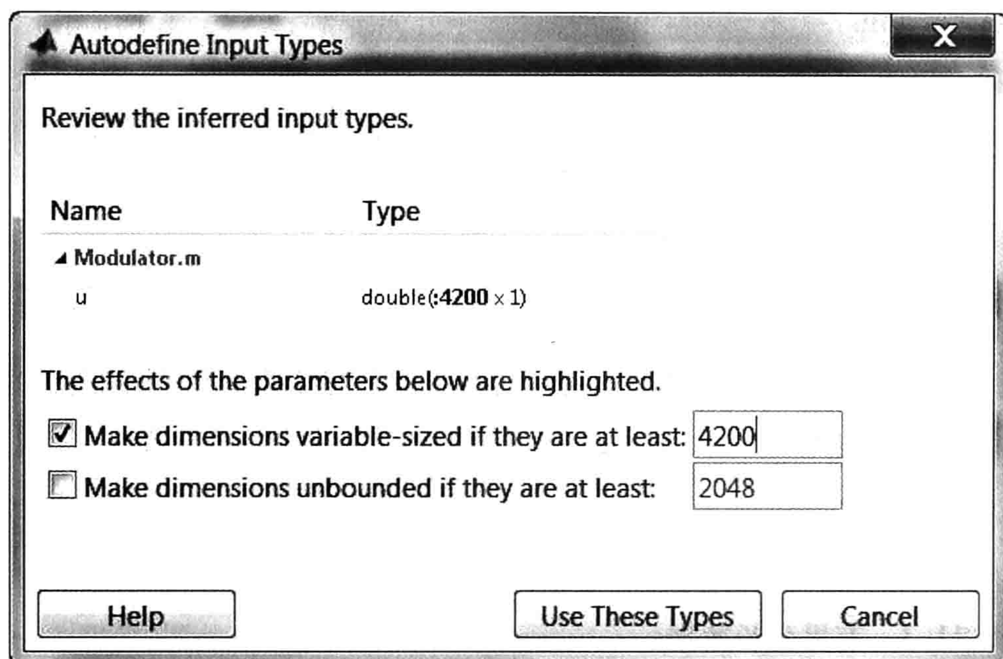


图 10.28　有界变长代码转换选项

Algorithm

MATLAB function

```
function y=Modulator_varsize_bounded(N)
assert(N<=2400);
persistent QPSK
if isempty(QPSK)
   QPSK        = comm.PSKModulator(4, 'BitInput', true, 'PhaseOffset', pi/4, ...
      'SymbolMapping', 'Custom', 'CustomSymbolMapping', [0 2 3 1]);
end
u=randi([0 1], N,1);
y=step(QPSK, u);
```

Modulator _ varsize _ bounded. m 函数只有一个输入（N），用来决定函数内每个变量的长度。因此，生成 Modulator _ varsize _ bounded 有界可变长度代码的 MATLAB 命令如下。

Algorithm

```
>> codegen -args {N} Modulator_varsize_bounded
```

```
>> u=randi([0 1], 4200, 1);
>> y=Modulator_mex(u);
>> whos u y
  Name          Size             Bytes  Class     Attributes

  u          4200x1              33600  double
  y          2100x1              33600  double    complex

>> u2=randi([0 1], 256, 1);
>> y=Modulator_mex(u2);
>> whos u y
  Name          Size             Bytes  Class     Attributes

  u          4200x1              33600  double
  y           128x1               2048  double    complex

>> u3=randi([0 1], 123456, 1);
>> y=Modulator_mex(u3);
MATLAB expression 'u' is not of the correct size: expected [:4200x1] found
[123456x1].

Error in Modulator_mex
```

图 10.29　调用调制器的有界可变长度 MEX 函数

10.11.4　无界变长数据

我们可以通过改变一个代码转换选项对调制器函数进行无界可变长度代码转换。在 MATLAB 代码转换工程中，我们只需要勾选"Make dimensions unbouded if they are at least（规定无界最少为）"，如图 10.30 所示。通过设定编辑栏中的最小值，我们可以规定代码转换器引擎将任何大于给定长度的变量按无界变量数据处理。因此，变量 u 的类型为 double（：inf x 1），意思为变量第一阶为无界。

codegen MATLAB 命令也可以支持无界输入长度生成 MEX 函数：

Algorithm

```
>> codegen Modulator -args {coder.typeof(0,[inf 1],1)}
```

为了验证其工作，我们运行上一节中的 MATLAB 脚本。我们可以看到，不论输入长度如何改变，MEX 都能产生正确的调制输出（见图 10.31）。

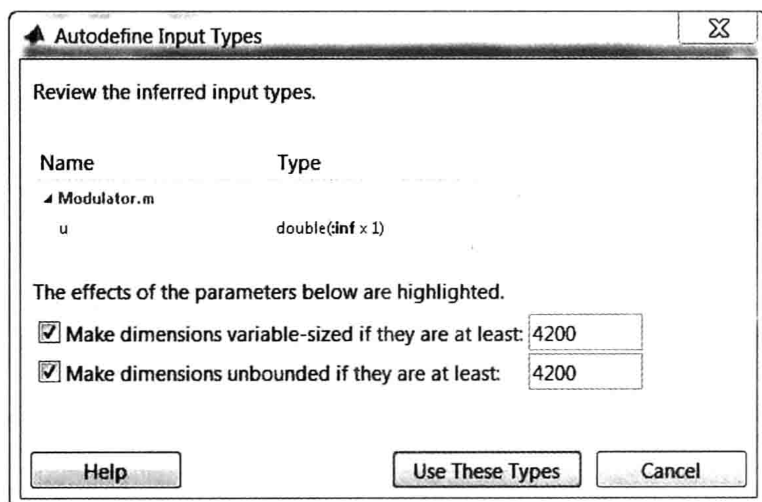


图 10.30　无界可变长度代码转换的选项

```
>> u=randi([0 1], 4200, 1);
>> y=Modulator_mex(u);
>> whos u y
  Name          Size              Bytes  Class     Attributes

  u          4200x1               33600  double
  y          2100x1               33600  double    complex

>> u2=randi([0 1], 256, 1);
>> y=Modulator_mex(u2);
>> whos u2 y
  Name         Size             Bytes  Class     Attributes

  u2         256x1              2048  double
  y          128x1              2048  double    complex

>> u3=randi([0 1], 123456, 1);
>> y=Modulator_mex(u3);
>> whos u3 y
  Name           Size              Bytes  Class     Attributes

  u3         123456x1             987648  double
  y           61728x1             987648  double    complex
```

图 10.31　调用调制器的无界可变长度 MEX 函数

10.12 集成外部C/C++代码

在本节中，我们将说明如何将MATLAB函数转换的C/C++代码与外部C/C++代码或C/C++开发环境集成。我们通过如下几步完成这一工作：

1）选择一个算法并以MATLAB函数表达。

2）生成MATLAB测试平台。脚本测试平台调用脚本设置不同的参数执行函数，记录每种情况的输出和耗时。执行测试脚本得到参考数值结果和参考运行时间。

3）生成函数的C代码。选择静态C库作为代码转换的输出类型。在目录中生成全部源文件和头文件（*.c和*.h文件）。

4）编辑C/C++主函数调用转换生成的C代码。

5）用一个简单的Makefile编译和连接C主函数和生成的函数C代码。其生成的可执行文件可在计算机上执行。可执行文件即C测试平台。

6）在MATLAB环境之外，运行生成的可执行文件（C测试脚本）。验证C测试脚本是否可以得到与参考结果相吻合的数值结果。最后，比较C脚本和MATLAB脚本在相同测试条件的耗时。

10.12.1 算法

我们首先选择一个算法，将其MATLAB代码与外部C代码集成。我们选择物理下行链路控制信道（PDCCH）的简化版处理算法[3]。在第九章中，我们考察过17种不同的PDCCH算法。在本节中，我们使用第九种算法，第八种算法的MEX函数使用通信系统工具箱中所有可用的系统对象。第九版的结果显示其使用并行多核处理的速度快于前八版。表示第八版算法的MATLAB函数如下：

Algorithm

MATLAB function

```
function [ber, bits]=zPDCCH_v8(EbNo, maxNumErrs, maxNumBits)
%% Constants
FRM=2048;
M=4; k=log2(M); codeRate=1/3;
snr = EbNo + 10*log10(k) + 10*log10(codeRate);
trellis=poly2trellis(7, [133 171 165]);
L=FRM+24;C=6; Index=[L+1:(3*L/2) (L/2+1):L];
%% Initializations
```

```
persistent Modulator AWGN DeModulator BitError ConvEncoder1 ConvEncoder2 Viterbi
CRCGen CRCDet
if isempty(Modulator)
   Modulator     = comm.QPSKModulator('BitInput',true);
   AWGN          = comm.AWGNChannel('NoiseMethod', 'Variance', 'VarianceSource',
'Input port');
   DeModulator =  comm.QPSKDemodulator('BitOutput',true);
   BitError       = comm.ErrorRate;
   ConvEncoder1=comm.ConvolutionalEncoder('TrellisStructure', trellis,
'FinalStateOutputPort', true, ...
      'TerminationMethod','Truncated');
   ConvEncoder2 =  comm.ConvolutionalEncoder('TerminationMethod','Truncated',
'InitialStateInputPort', true,...
      'TrellisStructure', trellis);
   Viterbi=comm.ViterbiDecoder('TrellisStructure', trellis,
'InputFormat','Hard','TerminationMethod','Truncated');
   CRCGen = comm.CRCGenerator('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
   CRCDet = comm.CRCDetector   ('Polynomial',[1 1 zeros(1, 16) 1 1 0 0 0 1 1]);
end
%% Processing loop modeling transmitter, channel model and receiver
numErrs = 0; numBits = 0; nS=0;
results=zeros(3,1);
while ((numErrs < maxNumErrs) && (numBits < maxNumBits))
   % Transmitter
   u          = randi([0 1], FRM,1);                     % Generate bit payload
   u1         = step(CRCGen, u);                          % CRC insertion
   u2         = u1((end-C+1):end);                       % Tail-biting convolutional coding
   [~, state]  = step(ConvEncoder1, u2);
   u3         = step(ConvEncoder2, u1,state);
   u4         = fcn_RateMatcher(u3, L, codeRate); % Rate matching
   u5         = fcn_Scrambler(u4, nS);                    % Scrambling
   u6         = step(Modulator, u5);                     % Modulation
   u7         = TransmitDiversityEncoderS(u6);            % MIMO Alamouti encoder
    % Channel
   [u8, h8]    = MIMOFadingChanS(u7);                     % MIMO fading channel
   noise_var = real(var(u8(:)))/(10.^(0.1*snr));
   u9         = step(AWGN, u8, noise_var);                % AWGN
   % Receiver
   uA         =  TransmitDiversityCombinerS(u9, h8);% MIMO Alamouti combiner
   uB         = step(DeModulator, uA);                    % Demodulation
   uC         = fcn_Descrambler(uB, nS);                  % Descrambling
   uD         = fcn_RateDematcher(uC, L);                 % Rate de-matching
   uE         = [uD;uD];                                  % Tail-biting
   uF         = step(Viterbi, uE);                        % Viterbi decoding
   uG          = uF(Index);
   y          = step(CRCDet, uG );                         % CRC detection
   results    = step(BitError, u, y);                     % Update number of bit errors
   numErrs    = results(2);
   numBits    = results(3);
```

```
  nS          = nS + 2; nS = mod(nS, 20);
end
%% Clean up & collect results
ber = results(1); bits= results(3);
reset(BitError);
```

为了能更方便的管理与C代码转换相关的文件和目录，我们创建一个新的目录并将所有MATLAB文件移动到里面。在本例中，我们建立的目录为C：\ Examples \ PDCCH，并将第八版算法所需的所有文件拷贝到里面。MATLAB脚本执行如下工作：

Algorithm

MATLAB script: MATLAB_testbench_directory

```
%% Create new directory in C:\ drive
PARENTDIR='C:\';
NEWDIR='Examples\PDCCH';
mkdir(PARENTDIR,NEWDIR);
%% Make that your destination directory
DESTDIR=fullfile(PARENTDIR,NEWDIR);
%% Copy 10 necessary files to destination directory
copyfile('Alamouti_DecoderS.m',DESTDIR);
copyfile('Alamouti_EncoderS.m',DESTDIR);
copyfile('fcn_Descrambler.m',DESTDIR);
copyfile('fcn_RateDematcher.m',DESTDIR);
copyfile('fcn_RateMatcher.m',DESTDIR);
copyfile('fcn_Scrambler.m',DESTDIR);
copyfile('MIMOFadingChanS.m',DESTDIR);
copyfile('TransmitDiversityCombinerS.m',DESTDIR);
copyfile('TransmitDiversityEncoderS.m',DESTDIR);
copyfile('zPDCCH_v8.m',DESTDIR);
%% Go to destination directory
cd(DESTDIR);
```

10.12.2 执行MATLAB测试平台

现在我们执行两个脚本：一个用于生成函数zPDCCH _ v8.m的MEX函数的编译脚本以及一个构成测试平台的调用脚本。在MATLAB函数存储的目录（C：\ Examples \ PDCCH）创建这两个脚本。我们用第一个脚本（MATLAB——build _ version9.m）生成第八版PDCCH算法的MEX函数。编译脚本的codegen命令如下：

Algorithm

MATLAB script: MATLAB_build_version9

```
MaxSNR=8;
MaxNumBits=1e7;
MaxNumErrs=MaxNumBits;
fprintf(1,'\nGenerating MEX function for PDCCH algorithm ...\n');
codegen -args { MaxSNR, MaxNumErrs, MaxNumBits} zPDCCH_v8 -o zPDCCH_v9
fprintf(1,'Done.\n\n');
MEX_FCN_NAME='zPDCCH_v9';
fprintf(1,'Output MEX function name:  %s \n',MEX_FCN_NAME);
```

测试平台（MATLAB _ testbench _ version9. m）遍历 Eb/N0 并记录对应的 BER 值。测试平台包含八种测试条件，对应从 0. 5 ~ 4. 0，以每 0. 5dB 为步长递增的 Eb/N0 值。我们计算并记录每个 Eb/N0 对应的 BER 值。仿真终止条件为处理比特数，由规定每个 Eb/N0 值处理中的最大误码数（MaxNumErrs）和最大比特数（MaxNumBits）确定。最后，我们记录仿真开始和结束时的系统时钟，并求它们的差得到八种测试条件的耗时。

Algorithm

MATLAB testbench: MATLAB_testbench_version9

```
MaxSNR=8;
MaxNumBits=1e7;
MaxNumErrs=1e7;
ber_vector=zeros(MaxSNR,1);
fprintf(1,'\nMATLAB testbench for PDCCH algorithm\n');
fprintf(1,'Maximum number of errors : %9d\n', MaxNumErrs);
fprintf(1,'Maximum number of bits   : %9d\n\n', MaxNumBits);
tic;
for snr=1:MaxSNR
   fprintf(1,'Iteration number %d\r',snr);
   EbNo=snr/2;
   ber= zPDCCH_v9(EbNo, MaxNumErrs, MaxNumBits);
   ber_vector(snr)=ber;
end
time_8=toc;
fprintf(1,'\nTime to complete %d iterations = %6.4f (sec)\n\n', MaxSNR, time_8);
for snr = 1:MaxSNR
   fprintf(1,'Iteration %2d  EbNo %3.1f  BER  %e\n', snr, snr/2, ber_vector(snr));
end
```

MATLAB 命令行窗口显示执行这个测试脚本的结果，如图 10. 32 所示。我们可以看到 MATLAB 测试脚本的参考 BER 值结果和仿真耗时。我们将在下面用 C 测试脚本得到的结果和这两个值做对比。

```
>> MATLAB_testbench_version9

MATLAB testbench for PDCCH algorithm
Maximum number of errors :  10000000
Maximum number of bits   :  10000000

Iteration number 1
Iteration number 2
Iteration number 3
Iteration number 4
Iteration number 5
Iteration number 6
Iteration number 7
Iteration number 8

Time to complete 8 iterations = 354.8422 (sec)

Iteration  1  EbNo 0.5  BER  5.311296e-03
Iteration  2  EbNo 1.0  BER  1.773132e-03
Iteration  3  EbNo 1.5  BER  5.100804e-04
Iteration  4  EbNo 2.0  BER  1.504942e-04
Iteration  5  EbNo 2.5  BER  3.509865e-05
Iteration  6  EbNo 3.0  BER  4.299835e-06
Iteration  7  EbNo 3.5  BER  8.999654e-07
Iteration  8  EbNo 4.0  BER  0.000000e+00
```

图 10.32 MATLAB 测试平台，列出参考输出值和耗时

10.12.3 生成C代码

现在，我们用codegen命令将zPDCCH _ v8. m函数转换为C代码。我们可以用codegen命令或MATLAB代码转换工程生成静态C库。当我们用codegen命令时，我们只需要定义配置选项lib，如下面的脚本所示：

Algorithm

MATLAB script: MATLAB_build_version9

```
MaxSNR=8;
MaxNumBits=2e6;
MaxNumErrs=MaxNumBits;
fprintf(1,'Generating source files (*.c) and header files (*.h) for PDCCH algorithm ...');
codegen -args { MaxSNR, MaxNumErrs, MaxNumBits} zPDCCH_v8 -config:lib -report
fprintf(1,'Done.');
FCN_NAME='zPDCCH_v8';
Location=fullfile(pwd,'codegen','lib',FCN_NAME);
fprintf(1,'All generated files are in the following directory: \n%s\n', Location);
```

完成之后，codegen 命令在 MATLAB 命令行输出指向生成的 C 代码的超链接。我们点击这个超链接，打开代码转换报告（见图 10.33）。

所有的 C 源文件和头文件都保存在总目录下独立的文件夹中。在本例中，总目录为 C：\ Examples \ PDCCH，所有源文件保存在子目录 codegen \ lib \ zPDCCH _ v8 中。图 10.34 中使用 ls 命令列出了所有生成的源文件和头文件。

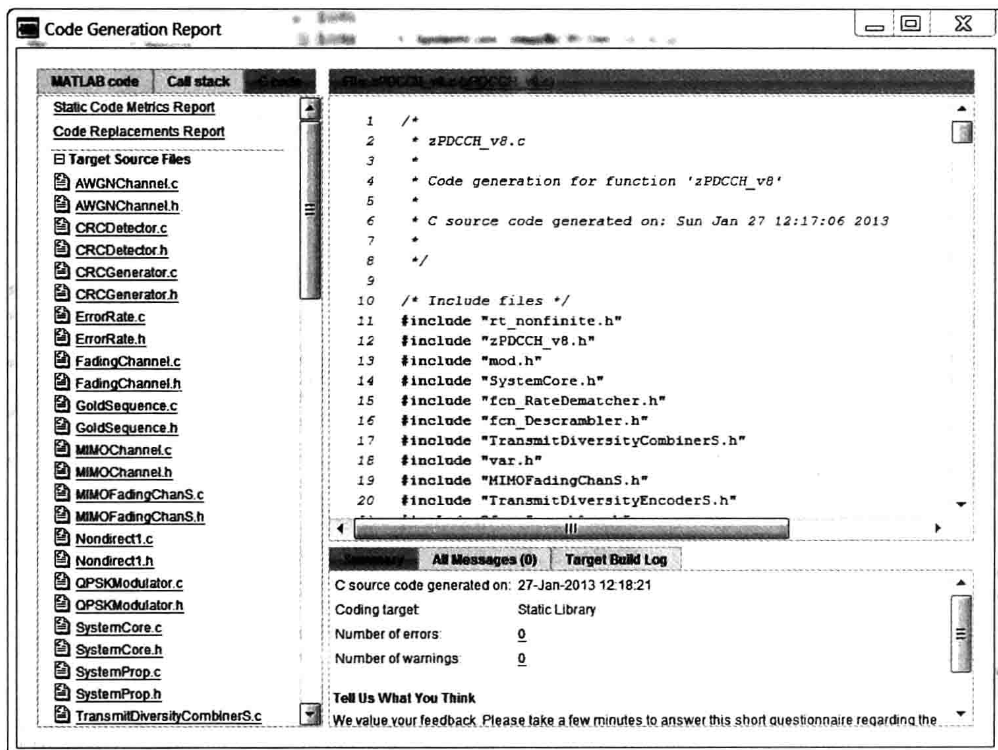


图 10.33 代码转换报告：显示 zPDCCH _ v8 算法转换的代码

10.12.4 接口函数 C 代码

在转换 MATLAB 函数为 C 代码后，剩余的开发工作可独立于 MATLAB 环境之外。为了生成 C 可执行文件，即 C 测试脚本，我们需要编写 C 主函数并在其中调用生成的接口函数。

在本例中，MATLAB 的接口函数为 zPDCCH _ v8. m。生成的 C 代码有三个头文件，分别定义接口 C 函数原型为：

1）主接口函数；

2）初始化函数；

3）终止函数。这些文件分别为 zPDCCH _ v8. h、zPDCCH _ v8 _ initialize. h 和 zPDCCH _ v8 _ terminate. h。

```
>> pwd

ans =

C:\Examples\PDCCH\codegen\lib\zPDCCH_v8

>> ls *.c

AWGNChannel.c        MIMOFadingChanS.c                 ViterbiDecoder.c        main.c
CRCDetector.c        Nondirect1.c                      diff.c                  mod.c
CRCGenerator.c       QPSKModulator.c                   fcn_Descrambler.c       permute.c
ErrorRate.c          SystemCore.c                      fcn_RateDematcher.c     rand.c
FadingChannel.c      SystemProp.c                      fcn_Scrambler.c         randn.c
GoldSequence.c       TransmitDiversityCombinerS.c      filter.c                repmat.c
MIMOChannel.c        TransmitDiversityEncoderS.c       floor.c                 rtGetInf.c

>> ls *.h

AWGNChannel.h        MIMOFadingChanS.h                 ViterbiDecoder.h        mod.h
CRCDetector.h        Nondirect1.h                      diff.h                  permute.h
CRCGenerator.h       QPSKModulator.h                   fcn_Descrambler.h       rand.h
ErrorRate.h          SystemCore.h                      fcn_RateDematcher.h     randn.h
FadingChannel.h      SystemProp.h                      fcn_Scrambler.h         repmat.h
GoldSequence.h       TransmitDiversityCombinerS.h      filter.h                rtGetInf.h
MIMOChannel.h        TransmitDiversityEncoderS.h       floor.h                 rtGetNaN.h
```

图10.34 生成的C源文件和头文件列表

Algorithm

C header file: *zPDCCH_v8.h*

```
/*
 * zPDCCH_v8.h
 *
 * Code generation for function 'zPDCCH_v8'
 *
#ifndef __ZPDCCH_V8_H__
#define __ZPDCCH_V8_H__
/* Include files */
#include <float.h>
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include "rt_nonfinite.h"
#include "rtwtypes.h"
#include "zPDCCH_v8_types.h"
/* Function Declarations */
extern void zPDCCH_v8(real_T EbNo, real_T maxNumErrs, real_T maxNumBits, real_T
*ber, real_T *bits);
#endif
/* end of code generation (zPDCCH_v8.h) */
```

Algorithm

C header file: *zPDCCH_v8_initialize.h*

```
/*
 * zPDCCH_v8_initialize.h
 *
 * Code generation for function 'zPDCCH_v8_initialize'
 *
 */
#ifndef __ZPDCCH_V8_INITIALIZE_H__
#define __ZPDCCH_V8_INITIALIZE_H__
/* Include files */
#include <float.h>
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include "rt_nonfinite.h"
#include "rtwtypes.h"
#include "zPDCCH_v8_types.h"
/* Function Declarations */
extern void zPDCCH_v8_initialize(void);
#endif
/* end of code generation (zPDCCH_v8_initialize.h) */
```

Algorithm

C header file: *zPDCCH_v8_terminate.h*

```
/*
 * zPDCCH_v8_terminate.h
 *
 * Code generation for function 'zPDCCH_v8_terminate'
 *
 */
#ifndef __ZPDCCH_V8_TERMINATE_H__
#define __ZPDCCH_V8_TERMINATE_H__
/* Include files */
#include <float.h>
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include "rt_nonfinite.h"
#include "rtwtypes.h"
#include "zPDCCH_v8_types.h"
/* Function Declarations */
extern void zPDCCH_v8_terminate(void);
#endif
/* end of code generation (zPDCCH_v8_terminate.h) */
```

C 主函数必须包含头文件。这对理解 C 主函数如何调用接口函数至关重要。通常一个算法需要：

1）一个初始化函数在处理循环外设定数据和参数；

2）一个主接口函数被处理循环调用；

3）终止函数清理初始化和接口函数使用的所有资源（数据、内存等）。下面的伪代码表述了主函数中 C 代码的结构以及调用接口函数的方法。

Algorithm

```
>> Initialization_function();
>> An iterative processing loop
  >> { that calls Main_entry_point_function many times;}
>> Terminate_function();
```

10.12.5　主函数 C 代码

下面的主函数 C 代码实际遵循了上一节中描述的调用结构。我们看到主函数 C 代码的前几行代码包含典型变量声明，并从命令行读取仿真数据。后面一部分的 C 代码为仿真关键部分。首先我们通过调用时钟函数记录仿真开始前的系统时间。然后我们调用 zPDCCH _ v8 _ initialize 函数在处理循环外对所需数据进行初始化。在处理循环内，我们调用主接口函数（zPDCCH _ v8）遍历 Eb/N0 值得到对应的 BER。最后，在处理循环结束后，我们调用 zPDCCH _ v8 _ terminate 函数释放初始化的数据资源，并再一次调用 clock 函数记录系统时间。仿真的总耗时为两次记录系统时间的差。在函数最后，我们输出耗时和 BER 结果。

Algorithm

C header file: ***zPDCCH_v8_terminate.h***

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include "rtwtypes.h"
#include "zPDCCH_v8.h"
#define MIN_EBNO 1
#define MAX_EBNO 9
int main( int argc, char *argv[])
{
 int EbNo, maxNumErrs, maxNumBits;
 double snr, elapsed;
 double ber_vector[MAX_EBNO], bits_vector[MAX_EBNO];
 time_t t1,t2;
 printf("\nMain C testbench for PDCCH algorithm\n");
```

```
  if ( argc!= 3) {
      printf("Usage : main.exe  Max_Number_of_Errors  Max_Number_of_Bits\n");
      exit(1);
    }
  maxNumBits = atoi(argv[1]);
  maxNumErrs = atoi(argv[2]);
  printf("Maximum Number of Errors :  %d\n", maxNumErrs);
  printf("Maximum Number of Bits   :  %d\n\n", maxNumBits);
  /*****************************************************/
  t1=clock();
  zPDCCH_v8_initialize();
  for (EbNo=MIN_EBNO; EbNo<MAX_EBNO; EbNo++)
  {
   printf("Iteration number %2d\n", EbNo);
   snr = 0.5*((double)EbNo);
   zPDCCH_v8(snr, maxNumErrs, maxNumBits, &ber_vector[EbNo], &bits_vector[EbNo]);
  }
  zPDCCH_v8_terminate();
  t2=clock();
  elapsed = ((double) (t2 - t1)) / CLOCKS_PER_SEC;
  /*****************************************************/
  printf("\nTime to complete %2d iterations = %f (sec) in C\n\n", (MAX_EBNO-MIN_EBNO),
elapsed);
  for (EbNo=MIN_EBNO; EbNo<MAX_EBNO; EbNo++)
     printf("Iteration %2d    EbNo: %3.1f   BER: %e\n", EbNo, 0.5*EbNo ,ber_vector[EbNo]);

  return(0);
} /* end of main() */
```

10.12.6 编译和连接

现在，我们已经将 C 主函数（main. c）添加到转换 MATLAB 算法生成的所有文件的目录中。我们同时需要一个简单的 MakeFile 编译和连接资源文件并得到可执行文件。Makefile 如图 10. 35 所示。它可以工作在安装有 Microsoft Windows 操作系统的 PC（台式机或笔记本）上。cl 为 C 编译器命令（Microsoft Visual C + + 编译器命令），并使用两个优化选项。通过一些简单更改，这个 Makefile 可以在 Linux 和其他 Unix 环境中工作，如 gcc 或其他编译器。Makefile 将源文件（ *. c）编译为对象文件（ *. obj），然后连接所有对象文件生成输出的可执行文件 main. exe。我们用 gmake（GUN Makefile 功能）调用 Makefile 并生成可执行文件。

图 10. 36 和 10. 37 所示为包含调用 Makefile 的步骤。首先，我们打开 Windows SDK7. 1 命令行，然后我们设定生成文件的目录，即 C 主文件，以及 Makefile 加载目录（C： \ Examples \ PDCCH \ codegen \ lib \ zPDCCH _ v8）。下面我

```
Makefile ×
# Run this makefile with gmake utility

CC = cl
CFLAGS = /O2 -I.
COMPILE = $(CC) $(CFLAGS) -c
OBJFILES := $(patsubst %.c,%.obj,$(wildcard *.c))

all: main.exe

main.exe: $(OBJFILES)
    $(CC) /O2 /Femain.exe $(OBJFILES)

%.obj: %.c
    $(COMPILE)  $<

clean:
    del *.obj main.exe
```

图 10.35 简单的 Makefile 以生成可执行文件（Microsoft Windows 版）

们调用 gmake 功能的 clean 选项移除所有对象文件和可执行文件。最后，通过调用 all 选项逐一编译所有资源文件并连接它们生成 main. exe 可执行文件。这个可执行文件为我们将要用来验证 PDCCH 算法代码转换结果是否正确的 C 测试脚本。

```
Setting SDK environment relative to C:\Program Files\Microsoft SDKs\Windows\v7.1
\.
Targeting Windows 7 x64 Debug

C:\Program Files\Microsoft SDKs\Windows\v7.1>cd C:\Examples\PDCCH\codegen\lib\zP
DCCH_v8

C:\Examples\PDCCH\codegen\lib\zPDCCH_v8>gmake -f Makefile -k clean
del *.obj main.exe

C:\Examples\PDCCH\codegen\lib\zPDCCH_v8>gmake -f Makefile -k all
cl /O2 -I.  -c  AWGNChannel.c
Microsoft (R) C/C++ Optimizing Compiler Version 16.00.40219.01 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

AWGNChannel.c
cl /O2 -I.  -c  CRCDetector.c
Microsoft (R) C/C++ Optimizing Compiler Version 16.00.40219.01 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

CRCDetector.c
cl /O2 -I.  -c  CRCGenerator.c
Microsoft (R) C/C++ Optimizing Compiler Version 16.00.40219.01 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.
```

图 10.36 执行 Makefile 的步骤：编译每个生成的 C 代码

```
zPDCCH_v8_rtwutil.c
cl /O2 -I.  -c  zPDCCH_v8_terminate.c
Microsoft (R) C/C++ Optimizing Compiler Version 16.00.40219.01 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

zPDCCH_v8_terminate.c
cl /O2 /Femain.exe AWGNChannel.obj CRCDetector.obj CRCGenerator.obj ErrorRate.ob
j FadingChannel.obj GoldSequence.obj MIMOChannel.obj MIMOFadingChanS.obj Nondire
ct1.obj QPSKModulator.obj SystemCore.obj SystemProp.obj TransmitDiversityCombine
rS.obj TransmitDiversityEncoderS.obj ViterbiDecoder.obj diff.obj fcn_Descrambler
.obj fcn_RateDematcher.obj fcn_Scrambler.obj filter.obj floor.obj main.obj mod.o
bj permute.obj rand.obj randn.obj repmat.obj rtGetInf.obj rtGetNaN.obj rt_nonfin
ite.obj setup.obj sum.obj var.obj xor.obj zPDCCH_v8.obj zPDCCH_v8_data.obj zPDCC
H_v8_emxutil.obj zPDCCH_v8_initialize.obj zPDCCH_v8_rtwutil.obj zPDCCH_v8_termin
ate.obj
Microsoft (R) C/C++ Optimizing Compiler Version 16.00.40219.01 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.

Microsoft (R) Incremental Linker Version 10.00.40219.01
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:main.exe
AWGNChannel.obj
CRCDetector.obj
```

图 10.37 执行 Makefile 的步骤：连接可执行文件

10.12.7 执行 C 测试平台

通过执行可执行文件（main. exe），如图 10.38 所示，我们实际上调用了算法的 C 测试脚本。为了进行有价值的比较，在命令行我们定义最大误码数和最大处理比特数的值与 MATLAB 测试脚本相同。

```
C:\Examples\PDCCH\codegen\lib\zPDCCH_v8>main.exe 10000000 10000000

Main C testbench for PDCCH algorithm
Maximum Number of Errors :  10000000
Maximum Number of Bits   :  10000000

Iteration number  1
Iteration number  2
Iteration number  3
Iteration number  4
Iteration number  5
Iteration number  6
Iteration number  7
Iteration number  8

Time to complete  8 iterations = 340.989000 (sec) in C

Iteration  1   EbNo: 0.5   BER: 5.311296e-003
Iteration  2   EbNo: 1.0   BER: 1.773132e-003
Iteration  3   EbNo: 1.5   BER: 5.100804e-004
Iteration  4   EbNo: 2.0   BER: 1.504942e-004
Iteration  5   EbNo: 2.5   BER: 3.509865e-005
Iteration  6   EbNo: 3.0   BER: 4.299835e-006
Iteration  7   EbNo: 3.5   BER: 8.999654e-007
Iteration  8   EbNo: 4.0   BER: 0.000000e+000

C:\Examples\PDCCH\codegen\lib\zPDCCH_v8>
```

图 10.38 C 测试平台输出一览

C测试平台遍历Eb/N0参数并相应的记录BER值。脚本记录完成八次循环所需的时间并输出每次循环得到的BER值。表10.2总结了遍历八个信噪比（SNR）值对1千万比特数据进行PDCCH处理所耗时间。

表10.2 PDCCH处理的MATLAB和C测试脚本仿真时间对比

PDCCH处理的仿真方法	仿真耗时/s
C脚本调用生成的C代码	339.96
MATLAB测试平台调用MEX函数	354.84

结果显示在本地C平台代码处理给定数量数据的C代码耗时，与MATLAB测试平台调用MEX函数进行相同处理耗时接近。这并不惊奇，因为任何MEX文件实际上都是由MATLAB代码转换器将MATLAB代码转换为C代码，编译并在MATLAB命令行调用。因此，MEX函数的性能应与手动将转换的C代码与外部C测试脚本集成的结果相当。

10.13 本章小结

在本章中，我们考察了使用MATLAB代码转换器将MATLAB代码转换为独立的C代码这一过程。我们首先回顾了多个C/C + +代码转换的用例，包括：

1）提高仿真速度；

2）独立可执行原型设计；

3）嵌入式实现；

4）与外部C项目或软件集成。

我们随后描述了两种代码转换方法：

1）在MATLAB命令栏调用codegen函数；

2）使用MATLAB代码转换器应用。

我们重点介绍了一些代码转换支持的语言特征和结构，并强调了特定的系统工具箱进行代码转换所支持的各种数据类型，包括定点型数据，和MATLAB程序使用的可变长度数据。最后，我们描述了将MATLAB算法转换的代码如何与外部C/C + +测试平台集成的流程。

参考文献

[1] MathWorks Product Releases, http://www.mathworks.com/support/compilers/R2013a/index.html (accessed 16 August 2013).

[2] MathWorks MATLAB Coder, http://www.mathworks.com/help/coder/index.html (accessed 16 August 2013).

[3] 3GPP Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation, Version 10.0.0 Release 10. TS 36.211.

11 总　　结

在本章中，我们对本书的要点进行总结并对未来工作提供一个框架。我们将这个总结分为四个部分。首先，我们回顾所研究的 LTE 收发系统建模的内容。随后我们总结系统建模仿真结果以及如何加速仿真。然后，我们叙述研究的如何将建模与实现相连接，以及如何用 C/C + + 软件进行原型仿真的内容。最后，我们回顾与 LTE PHY 层有关但没有在本书中详细阐述的要点。考虑到这些要点的重要性和篇幅，我们认为它们无法在本卷中进行详述。我们将它们留作未来的工作。

11.1　建模

作为本书第一个研究对象，我们提供了一个 LTE PHY 的数学建模概览。我们的目标是均衡的讨论每一个概念以加深理解。因此，我们决定以三个不同但互补的概念元素进行讨论：

1）提供一个 LTE 关键技术，如正交频分复用（OFDM）多载波传输和多输入多输出（MIMO）多天线方案的理论性概览；

2）对 LTE 协议进行技术性通览，特别关注下行链路传输的细节；

3）提供详细的 MATLAB 算法和测试平台以逐步学习和掌握 LTE 标准的仿真。

这种分立又均衡的讨论方式是本书的一个特点。

在本节中我们将总结对每个概念元素的讨论。

11.1.1　理论构思

在本书中，我们对 LTE 标准中关键技术的理论背景进行了讨论。我们研究了 LTE 多载波传输方案（如下行链路 OFDM 和上行链路的 SC – FDM（单载波频分复用））以及多天线 MIMO 传输方案。

我们解析了 MIMO – OFDM 传输技术理论基础的各方面内容。这些内容揭示了标准如何协同 MIMO 和 OFDM 工作完成移动通信高数据速率和高吞吐量的目标。我们同时讨论了 LTE 标准为提升整体性能从旧标准中吸收的有用技术，如链路自适应中的自适应性调制和编码以及高效的 Turbo 编码。考察 LTE 技术依托的理论背景也可以利于理解其他现代通信系统。如 OFDM 和 MIMO 技术同时也是 WiMAX 和新的无线 LAN 标准的基本技术。

11.1.2 标准规范

除了讨论理论基础之外，我们还对 PHY 信号处理进行了详细的阐述，特别关注了下行链路处理。我们回顾了标准中使用的若干个信道和信号。我们也对下行链路公共信道（DLSCH）处理和物理下行链路公共信道（PDSCH）处理进行了更深入的检视。

我们特别对 OFDM 和 SC - FDM 传输方案中使用的时 - 频资源网格构成进行了详细的考察。理解资源网格结构对理解 LTE 标准如何组织用户数据、控制信息、参考和导频信号，以及如何在接收端进行信道评估和均衡以复原数据有指导作用。它同时也展示了 LTE 标准如何轻松地将 OFDM 多载波方案和若干 MIMO 多天线技术集成。我们重点关注了标准对下行链路和各种上行链路传输模式，不仅仅是支持一个传输结构，而是提供了九种不同的传输模式。我们也阐述了这些不同的模式如何满足不同的调度和不同移动率和信道质量条件。

11.1.3 MATLAB 算法

本书的一个特点是我们在描述 PHY 模型的同时附带 MATLAB 和 Simulink 算法和脚本。我们希望提供 MATLAB 算法和脚本来为从事通信系统设计的 MATLAB 用户提供一个初始平台。我们希望提供一个未来社区成员协作的起点。在 MATLAB 和 Simulink 中仿真一个通信系统可执行协议，可以对系统设计先进算法起到不可估量的帮助。

我们在第 4 章中用 MATLAB 算法描述基本的绕码、调制和译码操作后，又陆续在第 5 章中介绍 OFDM 多载波传输和各种 MIMO 技术，包括第 6 章的发射分集和空分复用。在第 8 章中我们对链路自适应测率的 MATLAB 算法模型进行研究，并在第 8 章中架构一个 LTE 收发端模型涵盖下行链路前四种传输模式，然后对物理层仿真模型的质量和性能进行了各种评估。最后，在第 9 章和第 10 章我们给出了加速仿真的各种 MATLAB 算法，以及生成设计原型的独立程序 C 代码。这些要点将会在下面更详细的讨论。

11.1.3.1 接收端设计

如大多数通信标准，LTE 标准只定义了发射端操作。因接收端并没有单独定义，这为接收端算法创新提供了机遇。这些创新，通过网络设备和移动终端制造商的软硬件实现时，体现了每个移动通信系统供应商对专利和附加值的关注。

MATLAB 和它的通信系统设计工具为实现各种接收端组件设计提供了一个简单易用的环境。在本书中，我们对 LTE 系统模型中各种接收端组件的不同方案进行了说明。例如，在第 5 章中我们讨论的基于接受参考信号进行信道 - 频率响应估计的各种接收端操作。我们考察了理想信道估计器，和基于导频信号内插

的三种不同信道估计算法。内插函数对每个导频信号计算信道响应，从而得到整个资源网格的结果。又例如另一个例子，在第 6 章我们考察了各种 MIMO 接收端操作，研究了三种接收端计算最优发射符号估计的方法。这些技术基于了迫零（ZF）、最小均方误差（MMSE），和软判决球形译码器（SSD）算法。在第 8 章，我们考察了这些接收端算法对系统整体性能的影响。通过关注算法特征指标，如内存占用情况或计算复杂度，以及系统级指标，如比特误码率（BER）或吞吐量，我们可以对算法间的取舍进行评估。

11.1.3.2 仿真测试平台

在本书中，我们建立并更新了 MATLAB 测试平台以定量和定性的评估我们的 LTE 收发端性能。测试脚本包括发射端和接收端处理链，以及需要表现收发端情况的信道模型。这些脚本包括各种定性测量如频谱分析和星座图，以及定量测量如 BER 和吞吐量计算。

11.1.3.3 算法架构组件

选择合适的粒度对建模如 LTE 收发端这样的复杂系统的 MATLAB 算法至关重要。我们在本书中的系统建模和仿真反映了一个原则。我们不用重复设计通信系统架构组件，如调制器、卷积或 Turbo 编码器、译码器，或空－时区块编码组件等。例如，在实现 OFDM 发射器和接收器操作中，我们用 MATLAB 函数进行快速傅里叶变换（FFT）和快速傅里叶反变换（IFFT）。我们也可以用 DSP 系统工具箱的 dsp. FFT 和 dsp. IFFT 系统对象作为实现功能的另一种选择。这些系统对象可以处理定点型建模，它的组件大小不是 2 的幂。我们也使用了通信系统工具箱的调制器、turbo 编码器，和信道建模系统对象。通过这些工具箱中可用的组件，我们不需要花费时间在 MATLAB 中重新开发如 Turbo 译码器这样的基础组件，这可以帮助我们提升开发 LTE 收发端系统模型的速度。

11.2 仿真

一个全数学模型对开发任意一个通信系统标准很重要。但为了验证这样一个模型的精确性，我们必须进行一定数量有代表性的软件仿真。因为很多通信系统的性能指标，如吞吐量和 BER，都为概率性测量，需要依据大数据量处理。为了验证系统对抗外界劣化的能力，仿真需要足够规模已涵盖小概率状况。这些考虑迫使我们关注各种优化系统模型仿真速度的方法。我们考察了各种提高仿真速度的方法，并重点放在若干提高模型仿真速度的 MATLAB 和 Simulink 工具和技术上。

11.2.1　仿真加速

软件仿真加速体现了模型表达的易读性和性能优化的经典取舍。在我们一步一步开发 LTE 模型组件的过程中，我们做出了巨大牺牲以独立方式组织 MATLAB 代码。为了使代码易于理解，我们用若干个不太复杂的子组件组合成复杂的组件，而不使用任何捷径或代码模块。

为了提高仿真速度，我们有时需要吸收一些典型方法之长，包括模块化重复操作、合并处理循环，以及优化若干编译器或平台特征库的方法。在第 9 章中我们重点放在基于这些典型方法的各种 MATLAB 编程技术上。

第 9 章中一个最重要和突出的加速策略是在保持数值精度方面。在我们试验各种代码优化方法的过程中，我们的 MATLAB 代码执行速度越来越快，但它们都生成相同的数值结果。另一方面，因为接收端没有任何标准定义可供参考，因此更自由地尝试各种加速方法也对接收端设计有很大帮助。我们采取一种谨慎的方法进行代码优化并以保证数值精度为前提使处理更直观和易于验证。

11.2.2　加速方法

在第 9 章中，我们列举了若干 MATLAB 和 Simulink 中加速 LTE 系统模型仿真的技术。我们对控制信道处理模型进行了六种优化。这些技术既包括了优化 MATLAB 程序优化的方法，又包括应用更多的计算能力获得性能提升，以及将设计转换为编译过的 C 代码。我们从一个基准算法开始，通过一系列的分析和代码更新介绍了如下优化方法：

1）更好的 MATLAB 串行编程技术（向量化、预分配）；
2）使用系统对象；
3）MATLAB - C 代码转换（MATLAB 可执行文件，MEX）；
4）并行计算（parfor，spmd）；
5）GPU（图形处理单元）- 优化的系统对象；
6）Simulink 最快加速器模式进行仿真。

我们也展示了如何同时使用上述两种或更多的技术得到更快的仿真速度。一些技术的优势需要专用工具箱的功能才能实现。例如，MATLAB 并行计算可以发挥多核处理器、计算机束以及 GPU 的优势。MATLAB 代码转换器提供自动转换 MATLAB 代码为 C 代码的功能，使我们通过编译提高仿真速度。

11.2.3　实现

在建模和仿真的讨论之外，在第 10 章我们第一次讨论了 LTE 标准模型的实现。为了连接建模和实现，我们用 MATLAB 代码转换器生成 C 代码的模型原型。

我们展示了如何将 MATLAB 代码转换器生成的 ANSI/ISO C 源代码集成到外部 C/C + +测试平台和应用中。

11.3　未来工作的方向

在我们用 MATLAB 建立完善的 LTE 标准 PHY 层模型之前，还有大量的工作需要完成。在本书中，我们的工作更多体现在教学方面。我们关注 LTE 核心技术，将目标放在用户层面的信号处理方面。我们也根据需要设计了若干物理信号和信道、OFDM 资源网格中的数据组织，以及多天线技术处理。这些讨论阐明了传输基本技术并解释了实现高数据速率和提升系统吞吐量的可行性，如标准定义所述。

下一阶段的建模目标时提供一个软件方案作为参考，验证是否满足 LTE 标准的要求。为了满足标准，我们必须更具体的定义我们的仿真模型。最终的 LTE MATLAB 仿真模型需要通过所有标准测试并涵盖全部传输模式和情况。

下面我们将列举需要添加的建模组件，以升级我们的基准仿真模型达到如上要求。通过这些升级，我们的 LTE 系统模型可以在仿真中最终通过 LTE 标准适配测试。我们将分三个部分描述这些组件细节：用户层面建模、控制层面建模和系统接入建模。

11.3.1　用户层面

为了升级本书中开发的 LTE 仿真模型，我们需要首先对所有用户平面的属性进行建模。它们包括 FDD（频分双工）和 TDD（时分双工）两种时间帧、下行和上行链路公共信道的完整处理，以及 LTE - Advanced 标准的新特性。这些内容将在下一节讨论。

11.3.1.1　FDD 和 TDD 双工

如本书前文所述，LTE 标准定义了两种帧结构。第一种帧使用 FDD 模式而第二种帧为 TDD 模式。我们详细讲述了 FDD 和第一种帧结构。通过细微的更新，我们可以让 MATLAB 函数应用 TDD 双工模式的时间帧。与此类似，本书中我们使用普通循环前缀长度，而 MATLAB 代码可以通过一点点改动支持 OFDM 和 SC - FDM 传输的扩展循环前缀。

11.3.1.2　上行链路处理

在本书中我们的重点全放在下行链路传输中。未来的工作将涉及物理上行链路公共信道（PUSCH）的信号处理链。很多为下行链路传输开发的 MATLAB 组件可以不需要更改直接用于上行链路建模中。不过，两者的不同在于参考信号，上行链路使用基于 Zadoff - Chu 序列的参考信号。

11.3.1.3　完整的下行链路传输模式

我们详细考察了下行链路传输的前四种模式。完整的模型应该包括所有模式，包括下行链路增强型 MIMO 模式（模式 7、8，9）、UE（用户设备）特征波束赋形模式，以及信号 - 层空分复用模式。模型应包括各种参考信号的生成和分配，包括信道状态信息参考信号（CSI - RS）和解调参考信号（DM - RS）。

11.3.1.4　LTE - Advanced 特性

LTE MATLAB 接收端模型中也应包含 LTE - Advanced 的特征。它们集中存在于上行链路 MIMO 传输和载波聚合中。一个多用户上行链路 MIMO 实例以多 UE 可以在传输中共享资源的方式传输 PUSCH 子帧。这个技术可大幅提升上行链路吞吐量。载波聚合是另外一种 LTE - Advanced 特性，它可以覆盖多个载波进行下行链路传输。通过使用五个相邻载波的助力，载波聚合在实现 1Gbps 数据速率目标中发挥了主导作用。满足标准的 LTE PHY MATLAB 模型必须包含这两个特性。因为每个载波聚合带宽内的处理链彼此独立，并行处理可以明显提高其处理速度。因此，第 9 章中我们研究的加速技术可以直接应用于此。

11.3.2　控制层面处理

作为本书中的一个特点，我们关注了用户层面的公共信道处理。我们没有深入研究任何实现用户层面传输的控制信息。下行链路控制信息（DCI）和上行链路控制信息（UCI）必须加入 LTE MATLAB 系统模型的。

11.3.3　混合自动重传请求

LTE 标准定义了混合自动重传请求（HARQ）协议，以确保数据包传输的可靠性并管理偶发重传。在接受包无误情况下进行的新的传输。如果接受包有误，则需要重传。接收端为了连续提供数据包和缩短新数据的等待时间，我们可以按不同的 HARQ 处理号发送不同的数据包。在 LTE 下行链路协议中，DCI 格式包含与 HARQ 处理相关的内容。包括增量冗余版本和新数据指示。在本书中，我们的 MATLAB 函数没有涉及 HAQR 处理。在未来的工作中，这些处理将帮助控制过多的重传带来的系统延迟，同时 DLSCH 信道编码将包含 HARQ 信息。

11.3.4　系统接入模型

在本书中，我们关注于 UE 和 eNodeB（增强型节点基站）在连接已建立后之间的通信步骤和处理。LTE 标准为系统接入初始阶段、小区搜索，以及切换过程提供了众多组件、信号和功能。一个完备的 MATLAB 系统模型应该包括这些功能。本节中将特别阐述两个实例。

11.3.4.1 小区搜索和帧排序

下行链路传输信号的资源网格中实际上包含了移动单元系统接入、小区搜索和帧排序的信息。如前文所述，一些初始系统信息搭载在主信息块（MIB）上，并在给定调制和编码方案的资源网格中表示。MIB 包含系统带宽、系统帧序号（SFN），和物理混合 ARQ 指示信道（PHICH）配置信息。我们在第 5 章中研究了主同步信号（PSS）和辅助同步信号（SSS）以及物理广播信道（PBCH）（包含 MIB）。不过，我们没有描述编码和发射这些信息，或是使用这些信息得到系统初始带宽和其他关键信息的接收端操作算法。

11.3.4.2 随机接入

UE 使用物理随机接入信道（PRACH）发送一个报头用于初始化接入网络。因为这一过程为 UE 与 eNodeB 之间的第一次通信，系统并不知道 UE 设备的类型或协议。若干传输模式，如循环迟延分集（CDD）和预编码向量交换（PVS）等，为译码报头信息提供了一个临时的通道。因为我们并没有详细阐述过上行链路传输，故没有提供对初始系统接入的 MATLAB 算法和函数。

11.4 结语

在本章中，我们总结了本书的研究对象并为未来的研究提供了指引。我们将这些要点主要分为两类：建模和仿真。在建模中我们本着目标，均衡阐述了理解 LTE 标准有关的三个方面。我们对各个关键技术进行了理论和数学描述，按照标准定义，构建了 MATLAB 程序和测试平台，并结合仿真思想一步步实现我们的模型。在仿真部分中，我们重点关注提升仿真速度和有效利用软件构建如 LTE 这样的复杂系统。我们回顾了本书中涉及的若干仿真加速技术和原型机制。最后，我们列举了需要在未来工作中完善的其他技术要点，以期得到一个完备的 LTE 标准 PHY 层模型。

LTE 和 MATLAB 社区可能需要其他书籍完善我们的工作，以得到一个完整的 LTE 标准适配 MATLAB 模型。我们通过本书关注的关键技术和原理打下了坚实的基础。作者的下一本书将会面向标准适配性和完美涵盖全部标准协议。

译 后 记

——苍白的感想、肤浅的建议和殷切的希望

1. 我觉得工程师是一个非常需要专业技术的职业，并且工程师的实际工作确实是由一大堆单调的事儿组合而成的。国内外工程师在工作压力和工作内容上并没有什么非常大的区别，区别的只是单位劳动时间的薪资。当然你懂的，金钱换不来耐压能力。未来将要面对怎样的每天 8 小时（甚至 24 小时 + 节假日），这个是自己想要的样子，还是一种可以承受的牺牲——我觉得这是在决定献身于“工程师”这个头衔之前要思考的问题。

2. 工程师还是一个非常需要生活技术的职业。其技术不在于懂多少专业知识技能，而在于如何才能在避免自己的精神世界日渐空虚与忙碌（或许枯燥）的工作中找到平衡——许许多多前辈工程师一辈子在办公桌前、实验室里和外场机房披星戴月，身负过劳抑郁等各种身心亚健康，辛勤工作一辈子留下的，除了显赫的业界声望和无数的荣誉成就，或许还伴随着一身日益顽强的各种职业病后遗症以及开始逐渐落后的知识体系，个中的甘苦也许只有他们才能体会吧！我们这些后辈们，如何在这个岗位上经营好自己的人生——我觉得这是在决定献身于“工程师”这个头衔之后首先要思考的问题。

3. 多么具有事业心和正能量的工程师，也需要给自己找点业务之外的乐趣来调剂，吃也好宅也好，总是要给自己点喘息的时间。有没有这个时间，怎样才能安排出这个时间——我觉得这是在决定献身于“工程师”这个头衔之后第二个要思考的问题。

4. 真正的大牛工程师在业务以外也都不是菜鸟。只懂得 1 + 1 而对文学历史美术音乐电影乃至打游戏一无所知的工程师在如今的互联网时代肯定也难有太高的成就。大学里教了很多专业知识，但很少会教不属于本专业的知识——这点到了研究生以上尤其突出，且国内外都一样。等到独立生活之后会发现，很多时候需要的本领与常识和自己本专业没有一丁点儿关系。

5. 少迷信创新。市面上的新东西绝大多数都是改改已有的成果的旧货而来的，只有极少数是全新的，这极少数的全新技术基本都是领域内凤毛麟角的顶级大牛的偶然奉献。想成为这样的顶级大牛，得先做好第 4 条。

6. 珍爱生命，多注意身心健康。工程师不容易，多想想自己的朋友家人。多喝点水，少着点急。

以上就是我干了这么些年工程师的一点感想和建议，主要面对即将走入社会的理工科学生们。

最后一些话来总结一下本书和 MATLAB。本书主要说了这么三件事：

1. 介绍了 MATLAB 系统对象和若干个与通信数字处理有关的工具箱。

2. 对 LTE 系统基本构架进行了建模，让我们对 LTE 标准里面有什么有了初步的了解。

3. 告诉我们算法原型搭建可以靠 MATLAB，追求最终系统实现方案的性能指标还是要靠 C/C + +。

对于 MATLAB，我想说它确实是算法设计层面很优秀和给力的工具。另一方面，广大工程师们还是要学好其他底层软硬件工具和知识，来干剩下的工作——这些内容比 MATLAB 能做的多太多了。

还有一点补充，在我们每个人并不短暂的人生中，找到爹妈之外爱你的人（男人/女人）比当好工程师重要得多，也要难得多。共勉！

——武冀，2015 年春于香格里拉

国际视野　科技前沿

本书是LTE技术图书中的一块瑰宝，这是众多读者的评论。本书具有三个优势，即大量的MATLAB实例和测试平台源代码，深入浅出讲解LTE物理层规范及重点讲解LTE内容和演变过程。可让读者通过MATLAB程序迅速掌握LTE技术。

—— MathWorks美国总部产品市场经理　赵志宏（John Zhao）

本书使用MATLAB清晰地解释了LTE技术的数学思想与架构，详细解读了LTE移动通信标准，并聚焦其物理层、构建算法和测试平台等，通过仿真帮助读者从另一个角度更加深入理解LTE技术标准和关键技术。

—— 中国移动通信集团设计院有限公司 李楠

本书为使用MATLAB进行通信仿真的工作人员和研究者们提供了莫大的帮助。LTE这样一个先进的技术，通过这本书真正意义上得以从理论到实践，启迪中国的4G通信开发者们。欢迎大家来MATLAB中文论坛讨论LTE技术。

—— MATLAB中文论坛（www.iLoveMATLAB.cn）

国际信息工程先进技术译丛

《全面详解LTE：MATLAB建模、仿真与实现》
《低速无线个域网：实现基于IEEE 802.15.4的无线传感器网络（原书第3版）》
《6LoWPAN：无线嵌入式物联网》
《虚拟网络——下一代互联网的多元化方法》
《Android系统安全与攻防》
《移动无线信道》（原书第2版）
《LTE-Advanced：面向IMT-Advanced的3GPP解决方案》
《认知无线电通信与组网：原理与应用》
《LTE/SAE网络部署实用指南》
《IP地址管理原理与实践》
《自组织网络：GSM，UMTS和LTE的自规划、自优化和自愈合》
《实现吉比特传输的60GHz无线通信技术》
《LTE自组织网络（SON）：高效的网络管理自动化》
《UMTS中的LTE：向LTE-Advanced演进》（原书第2版）
《UMTS中的WCDMA - HSPA演进及LTE》（原书第5版）
《UMTS中的LTE:基于OFDMA和SC-FDMA的无线接入》
《基于4G系统的移动服务技术》
《UMTS蜂窝系统的Q0S与Q0E管理》
《UMTS-HSDPA系统的TCP性能》
《基于射频工程的UMTS空中接口设计与网络运行》
《基于蜂窝系统的IMS—融合电信领域的V0IP演进》



机械工业出版社微信服务号

WILEY

上架指导　工业技术／移动通信



ISBN 978-7-111-48919-1　定价：88.00元

[General Information]
书名=全面详解LTE：MATLAB建模、仿真与实现
SS号=13720567